MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

Framework de controle de acesos baseado em gamificação

RELATÓRIO FINAL DE PROJETO DE INICIAÇÃO CIENTÍFICA
(PIBIC/INPE/CNPq)

Weslei Luiz de Paula Pinto (Faculdade de Tecnologia de São José dos Campos, Bolsista PIBIC/CNPq)
E-mail: weslei.paula@fatec.sp.gov.br

Eduardo Martins Guerra (COCTE/LABAC/INPE, Orientador)
E-mail: guerraem@gmail.com

São José dos Campos
2019

Sumário

Listas de Figuras

Lista de Tabelas

Lista de Símbolos

<T> Convenção de declaração de tipo genérico

<?> Declaração de caractere coringa, do inglês *wildcard*. Utilizado para quando não se sabe o tipo que será recebido

Lista de Abreviaturas

ABAC *Attribute Based Access Control*

API *Application Programming Interface*

DSM *Dependency Structure Matrix*

INPE *Instituto Nacional de Pesquisas Espaciais*

JVM *Java Virual Machine*

JRE *Java Runtime Environment*

MAC *Mandatory access control*

PD *Proxy Dinâmico*

POM *Project Object Model*

RBAC *Role Based Access Control*

SQL *Structured Query Language*

TDD *Test Driven Development*

Resumo

Geralmente, jogos que possuem restrições têm recursos desbloqueados conforme o progresso do personagem ou a consecução de algum objetivo. Aplicações que utilizam mecanismos de gamificação como pontos, *rankings*, troféus e *rewards* para condicionar comportamentos em seus usuários, tendem a utilizar a abordagem de controle de acesso para aumentar o valor das recompensas recebidas a partir da liberação de autorização conforme a utilização do usuário, desbloqueando novos recursos ou funcionalidades. O Esfinge Gamification é um *framework* que implementa mecanismos de gamificação, para simplificar o desenvolvimento de *softwares* que utilizem estas características. Atualmente o Esfinge Gamification não possui funcionalidades de controle de acesso, portanto, este trabalho é motivado pela oportunidade da criação de funcionalidades de controle de acesso baseadas em conquistas existentes no *framework*. Para isto o Esfinge Guardian foi integrado ao Esfinge Gamification, possibilitando a utilização de mecanismos de autorização em aplicações Java a partir da definição de metadados via anotações, para assim, desencadear comportamentos de autorização transparentes na aplicação.

1 INTRODUÇÃO

Gamificação é a utilização de mecanismos de jogos em aplicações com outros contextos, estes mecanismos são: ganho de pontos, *ranking*, troféus e recompensas, assim como processos que devem ser concluídos, e, quando são finalizados com sucesso, algum tipo de conquista é recebida, aumentando a motivação e o interesse dos usuários (ROBSON et al., 2015). *Softwares* utilizam esta ideia para estimular usuários a realizar determinado processo ou continuarem utilizando seus serviços, aplicando lógicas que dão conquistas quando forem finalizadas, desta forma incentivam seus usuários a continuar utilizando a solução criada (HAMARI; KOIVISTO; SARSA, 2014). A Figura 1 exibe o crescente interesse pelo termo *gamification* nos últimos anos (GROH, 2012), devido a este crescimento, aplicações com contextos não relacionados a jogos vem adotando a ideia dos *games* em suas soluções, alguns exemplos são: Duolingo, aplicativo para aprender idiomas e Foursquare, aplicativo para verificar classificações de locais.

Figura 1 - Consultas sobre a palavra *gamification* durante os anos

Fonte - Origem de dados: Google Trends Google (2019)

Estas aplicações possuem fins totalmente diferentes, porém, aplicam lógicas semelhantes de gamificação, que é a continuidade da utilização para a obtenção de mais conquistas. As conquistas são chamadas de ofensivas no Duolingo e de pontos no Foursquare. Baseado nisto, a gamificação é útil em diversas aplicações, porém, é necessário um contexto para que estes padrões citados anteriormente (pontos, rankings, troféus e recompensas) sejam aplicados para cativar e/ou estimular os usuários, o que pode não ser um processo simples dependendo do objetivo da aplicação. O *framework* Esfinge Gamification disponibiliza uma solução para o problema citado, abstraindo a complexidade da criação destas funcionalidades. Este tem como objetivo desacoplar qualquer lógica de gamificação da aplicação que utilizar suas funcionalidades, permitindo que o programador não se preocupe em como criá-las no código que está desenvolvendo, apenas inserir anotações do *framework* onde precisar de comportamentos de gamificação Guerra et al. (2011a) e estes serão desencadeados conforme Figura 2, onde o método "answerQuestion()" (linha 5) possui a anotação @PointsToUser, esta anotação atribui 10 pontos do tipo ANSWER ao usuário atual configurado no *framework*, porém, toda a lógica para a adição destes pontos é realizada pelo Esfinge Gamification, isto é transparente na aplicação.

Figura 2 - Exemplo de utilização do Esfinge Gamification

```
1
2  // Interface com anotacao do framework
3  public interface Questionnaire {
4      @PointsToUser(name = "ANSWER", quantity = 10)
5      public void answerQuestion(String answer);
6  }
7
8  public class QuestionnaireImpl implements Questionnaire {
9      // Implementacao omitida
10 }
11
12 public static void main(String[] args) {
13     // Configuracao do framework omitida
14     Questionnaire q = (Questionnaire) GameProxy.createProxy(new
           QuestionnaireImpl());
15     q.answerQuestion("Gamification works!");
16 }
```

Fonte - Adaptado de Guerra et al. (2011a)

Adicionar restrições de uso aumentam o empenho dos usuários a fim de conseguir novas funcionalidades ou recursos que ficam disponíveis conforme a utilização, isto é reconhecido na área da Psicologia Comportamental como reforço positivo (SKINNER, 1990). Geralmente, jogos que possuem restrições têm os recursos desbloqueados conforme o progresso do personagem ou a consecução de algum objetivo, aplicações que implementam lógicas de gamificação costumam possuir esta abordagem para atingir os objetivos citados anteriormente a partir de regras de autorização. Atualmente o Esfinge Gamification não possui funcionalidades deste tipo, portanto, este trabalho é motivado pela oportunidade da criação de funcionalidades de controle de acesso baseadas em conquistas existentes no *framework*, para que, caso mecanismos de autorização sejam necessários em aplicações que utilizam a solução, estes poderão ser adicionados por meio de anotações, e assim, desencadear comportamentos de autorização transparentes a aplicação, seguindo a filosofia dos projetos Esfinge.

1.1 Problema

O problema identificado foi a falta de opções para adicionar regras de autorização em aplicações que utilizem o Esfinge Gamification a partir dos dados de gamificação gerados pelo *framework*.

1.2 Objetivo Geral

O objetivo deste trabalho é prover mecanismos de autorização baseados em conquistas para o Esfinge Gamification, ou seja, é a criação de funcionalidades de controle de acesso a partir dos dados de gamificação disponibilizados pelo *framework*.

1.3 Objetivo Específico

Para a realização deste objetivo foram estabelecidas metas específicas:

- Integração entre os *frameworks* Esfinge Gamification e Esfinge Guardian, criando componentes de autorização para o Esfinge Gamification;
- Criação de anotações para obter controle de acesso baseado em conquistas, utilizando os recursos disponibilizados pelo *framework*.

1.4 Metodologia

O desenvolvimento dos componentes adicionais para o *framework* tem como etapa inicial a integração entre o Esfinge Guardian e o Esfinge Gamification, após isto, a criação de anotações para aumentar a abstração das funcionalidades de autorização oferecidas pela solução, realizando assim uma interface amigável, e permitindo ao desenvolvedor aplicar regras de autorização de uma maneira simples e eficaz. Para realizar estes objetivos, técnicas de reflexão e anotação em java foram utilizadas.

1.5 Organização do trabalho

Este Trabalho está organizado nos seguintes capítulos:

- Capítulo 2: Contém a fundamentação teórica necessária para entender os assuntos abordados no capítulo de desenvolvimento;
- Capítulo 3: Possui o desenvolvimento do trabalho, como o objetivo foi concretizado;
- Capítulo 4: Neste capítulo o desenvolvimento é testado e sua teoria validada com um estudo de caso;
- Capítulo 5: Aborda as conclusões e considerações finais a respeito deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

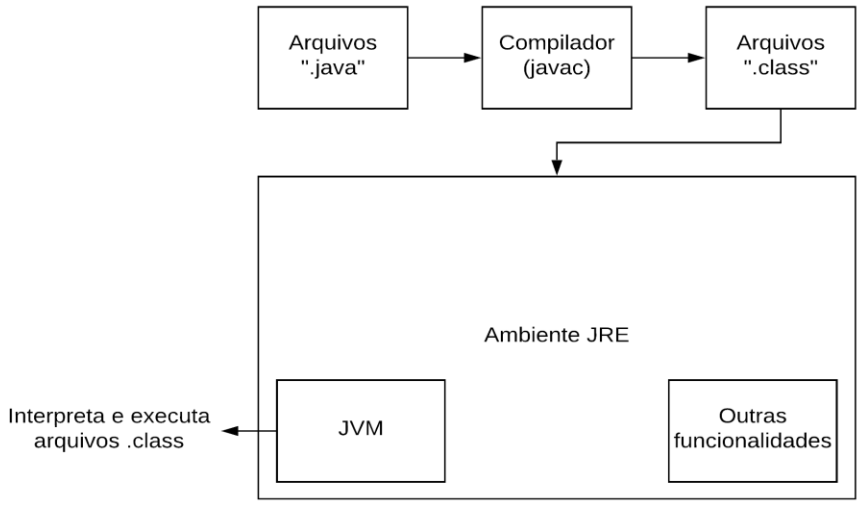
Neste capítulo serão fundamentados os conhecimentos básicos para o entendimento do trabalho.

2.1 Java

Java é uma linguagem de programação multiplataforma, concorrente (executa mais de uma tarefa em paralelo), baseada em classes e orientada a objetos (JOY et al., 2000).

A linguagem Java é compilada e interpretada. Após escrever programas em Java, estes são salvos como código fonte com extensão ".java". Quando estes códigos fontes são compilados, um arquivo binário chamado de arquivo de classe com extensão ".class" é gerado. Estes arquivos não são executados diretamente pelos processadores, pois eles não contêm instruções para eles. Os programas Java são compilados em um formato de arquivo chamado *bytecode*. Desta forma, esses programas podem ser executados em qualquer sistema operacional que possua um interpretador JVM (Java *Virtual Machine*) em um JRE (Java *Runtime Environment*) conforme Figura 3.

Figura 3 - Ambiente Java



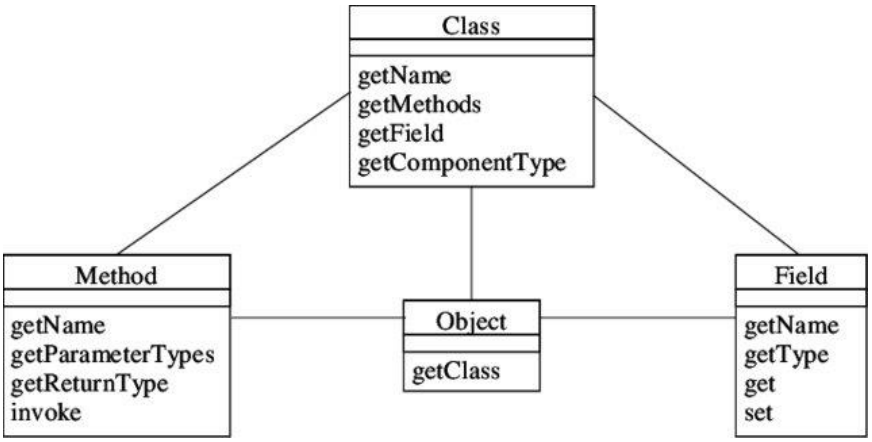
Fonte - Adaptado de Oracle (2010)

Assim, o código precisa ser compilado apenas uma vez para funcionar em qualquer sistema operacional que possua a configuração citada, pois os *bytecodes* serão executados da mesma forma pela JVM (ARNOLD; GOSLING; HOLMES, 2005). É também uma linguagem fortemente tipada, isto é, as características das variáveis têm que ser definidas em tempo de compilação. Ela possui um coletor de lixo (*garbage collector*) para evitar problemas de segurança como *deadlock* (JOY et al., 2000).

2.1.1 Reflexão

De acordo com Guerra (2014) o conceito de reflexão pode ser definido como um processo em que um programa pode visualizar e alterar sua própria estrutura ou comportamento. As classes de reflexão disponíveis em Java localizam-se no pacote java.lang.reflect, porém, na *Application Programming Interface* (API) padrão do Java as classes deste pacote aplicam o conceito de introspecção, que é a obtenção de informações sobre sua estrutura, sem possibilidade de modificação, não existem funcionalidades de modificação disponibilizadas por padrão pela API Reflection. Com o conceito de introspecção é possível recuperar informações de acordo com o tipo atual, que são: objeto, método, anotação, interface, classe ou campo. A Figura 4 exibe os principais métodos dos tipos classe, método, objeto e atributo.

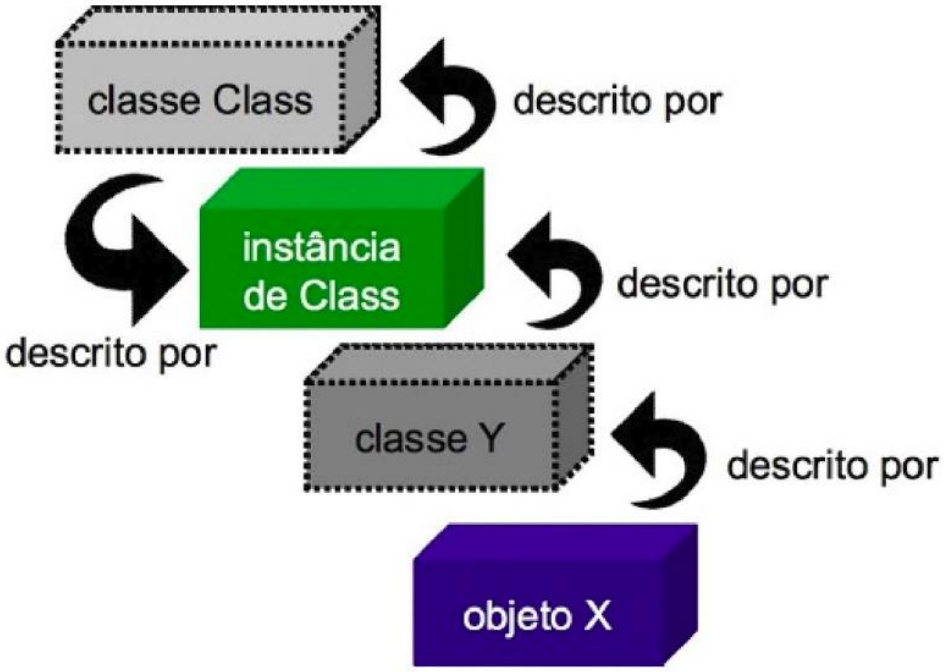
Figura 4 - Principais métodos de introspecção disponíveis por tipo



Fonte - Parson (2000)

Cada tipo citado anteriormente possui sua implementação para retornar as informações necessárias, a classe Class possui as características da classe do objeto atual em memória, e a partir dela é possível realizar as operações de introspecção vide Figura 5, isto é, a classe Y possui uma instância de Class com seus metadados como seu nome, construtores, assinatura dos métodos, nome e modificadores dos atributos, existência de anotações, e assim sucessivamente até que a classe Class possua uma instância com suas características.

Figura 5 - Fluxo de reflexão para obter informações sobre instâncias



Fonte - Guerra (2014)

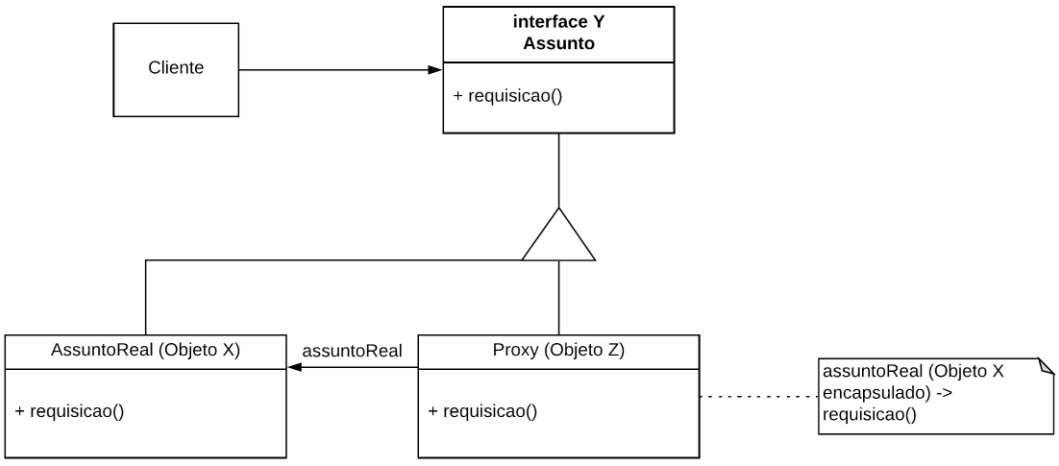
Com reflexão é possível recuperar informações padrões sobre os dados (classes, métodos e atributos), porém não é possível realizar processos mais sofisticados como validações ou alguma regra de negócio baseando-se nisto apenas (GUERRA; FERNANDES; SILVEIRA, 2010), para este tipo de necessidade foi criada uma forma de adição de metadados em Java, para facilitar este processo que era realizado via arquivos ou recursos externos, esta solução foi a criação das anotações (BUCKLEY, 2002).

2.1.1.1 *Proxy* Dinâmico

Para entender o conceito de *proxy* dinâmico (PD), o padrão de projeto *proxy* proposto por Gamma (1995), exibido na Figura 6 precisa ser abordado antes. Este tem como objetivo o encapsulamento de um objeto e a implementação de sua interface, ou seja, um objeto X com uma interface Y é encapsulado por um objeto Z que implementa a interface Y também, possuindo os mesmos métodos.

Quando o cliente (Figura 6) realiza chamadas de métodos do assuntoReal (objeto encapsulado), estas são enviadas primeiramente ao *proxy*, portanto, todas as execuções de métodos do objeto encapsulado são interceptadas pelo *proxy*. Na execução do *proxy* qualquer lógica pode ser aplicada, como validações ou chamadas de outros métodos. Ao final da execução das lógicas definidas no *proxy*, a responsabilidade é devolvida ao objeto encapsulado. Um problema na utilização de *proxies* é a dependência da interface, pois caso outro objeto precise ser encapsulado com uma interface diferente da suportada pelo *proxy* sua interface também precisa ser implementada, ou seja, outro *proxy* deverá ser criado.

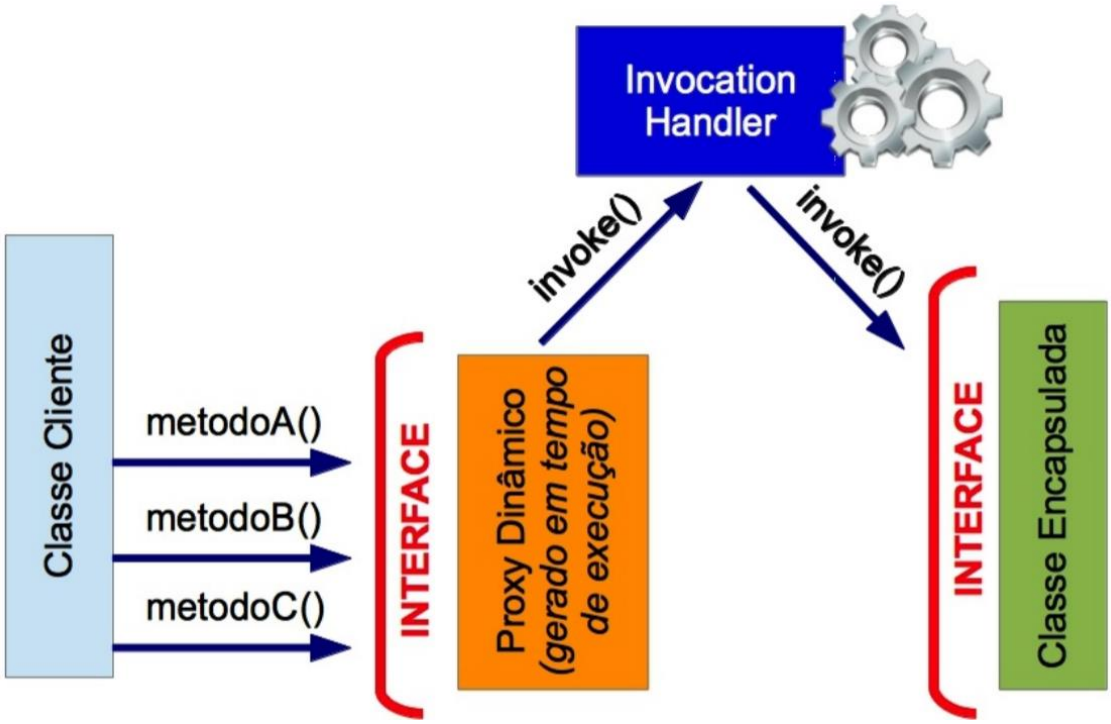
Figura 6 - Padrão de projeto proxy



Fonte - Adaptado de Gamma (1995)

PD é um recurso disponibilizado pela API reflexão (*reflection*) do Java, a fim de interceptar invocações de objetos de forma semelhante ao padrão de projeto *proxy* apresentado na Figura 6, entretanto sua principal diferença é que objetos com qualquer interface podem ser interceptados, devido a forma de criação do PD pela JVM a partir do método estático "newProxyInstance()" da classe Proxy, que devolve uma instância que será utilizada no lugar do objeto encapsulado com os parâmetros passados para este método, a Figura 7 apresenta o funcionamento de um PD.

Figura 7 - Funcionamento do proxy dinâmico



Fonte – Guerra (2014)

O método estático criaProxyDinamico() da Figura 8 recebe um objeto como parâmetro (linha 8), e ele é encapsulado de acordo com a ideia do padrão de projeto *proxy* na linha 12, a partir da criação de uma nova instância da classe atual (ProxyDinamicoExemplo). Com essa nova instância, o método construtor privado localizado na linha 4 é invocado, e nele, a referência do objeto recebido na linha 8 é encapsulada.

O retorno do método criaProxyDinamico() é um PD com objeto o encapsulado a partir dos parâmetros passados para o método estático newProxyInstance(). Este recebe três parâmetros: 1) ClassLoader do objeto; 2) Interfaces do objeto; 3) Uma instância da implementação de InvocationHandler; O PD é criado com esses parâmetros pela JVM conforme já dito, e todas invocações do objeto são interceptadas pelo terceiro parâmetro passado na instância do *proxy*, que é uma implementação de InvocationHandler, neste exemplo a própria classe ProxyDinamicoExemplo.

Figura 8 - Criação do proxy dinâmico

```
1  public class ProxyDinamicoExemplo implements InvocationHandler {
2          private Object objetoEncapsulado;
3
4      private ProxyDinamicoExemplo(Object objetoEncapsulado) {
5          this.objetoEncapsulado = objetoEncapsulado;
6      }
7  //Recupera ClassLoader e interfaces do objeto passado como parametro
       via reflexao
8      public static <T> T criaProxyDinamico(Object object) {
9          return (T) Proxy.newProxyInstance(
10                 object.getClass().getClassLoader(),
11                 object.getClass().getInterfaces(),
12                 new ProxyDinamicoExemplo(object)
13         );
14     }
15 //Implementacao do metodo Invoker omitido
16 }
```

Fonte - Produção do autor

A interface InvocationHandler do pacote java.lang.reflect define um método apenas, o Invoke, vide Figura 9, este método é executado antes de qualquer invocação do objeto encapsulado no processo anterior de criação do *proxy*.

A cada invocação do objeto encapsulado, o método invoke() é executado antes, de acordo com a Figura 10 onde é possível verificar a criação do PD na linha 4, onde o método criaProxyDinamico() da Figura 8 é chamado, retornando um PD do tipo Executor, este possui o método executionTime() que retorna uma instância de java.util.Date.

Figura 9 - Implementação da interface InvocationHanlder

```
1  public class ProxyDinamicoExemplo implements InvocationHandler {
2  //encapsulamento do objeto e metodo de criacao do proxy omitidos
3      @Override
4      public Object invoke(Object proxy, Method method, Object[] args)
           throws Throwable {
5          Class clazz = this.getClass();
6          Logger.getLogger(clazz.getName())
7              .log(
8                          Level.INFO, "Metodo " +
9                          method.getName() +
10                         " interceptado pelo proxy dinamico " +
11                         clazz.getSimpleName()
12             );
13         Object resultadoDaInvocacao = method.invoke(objetoEncapsulado,
               args);
14         return resultadoDaInvocacao;
15     }
16 }
```

Fonte - Produção do autor

Figura 10 - Registros da invocação interceptada

```
1  public class App {
2      public static void main(String[] args) {
3
4          Executor executor = ProxyDinamicoExemplo.criaProxyDinamico(new
               ExecutandoProxyDinamico());
5          System.out.println(executor.executionTime());
6      }
7  }
```

```
Console
<terminated> App [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (25 de abr de 2019 20:24:47)
abr 25, 2019 8:24:49 PM br.fatec.proxy.ProxyDinamicoExemplo invoke
INFO: Metodo executionTime interceptado pelo proxy dinamico ProxyDinamicoExemplo
Thu Apr 25 20:25:50 BRT 2019
```

Fonte - Produção do autor

Quando o PD é invocado, *logs* são criados conforme linha 6 da Figura 9, esta linha exibe o método interceptado e nome do PD. Sempre que o objeto executor for invocado, um *log* será gerado.

2.1.2 Anotação

Anotações em Java são utilizadas para definição de metadados, isto é, informações sobre algo, estas podem ser inseridas em: Pacotes; Classes; Interfaces; Métodos; Atributos; Parâmetros; Construtores; e em Anotações (DARCY, 2005). O local onde uma anotação pode ser inserida é definido pela anotação @Target que recebe de um a N parâmetros do Enum java.lang.annotation.ElementType[1], a Tabela 1 exibe os parâmetros disponíveis e seu escopo. Quando uma anotação é inserida fora de seu escopo um erro de compilação é exibido alertando que sua utilização está incorreta (JOY et al., 2000).

Tabela 1 - Parâmetros e restrições de anotações

Parâmetro	Escopo
TYPE	Classes, Enums, Anotações, Interfaces e Pacotes
FIELD	Atributos
METHOD	Métodos
PARAMETER	Parâmetros
CONSTRUCTOR	Construtores
LOCAL_VARIABLE	Variáveis locais (dentro de métodos)
ANNOTATION_TYPE	Anotações
PACKAGE	Pacotes
TYPE_PARAMETER	Tipos genéricos definidos em Métodos, Classes e Interfaces
TYPE_USE	Variáveis com tipos genéricos

Fonte - Adaptado de Joy et al. (2000)

Definido o escopo de atuação da anotação é preciso definir seu escopo de duração, isto é, até onde ela será mantida no código, pois as anotações não são persistidas até o tempo de execução sem uma anotação chamada @Retention. Esta define o ciclo de vida da anotação, na Tabela 2 seus parâmetros, que são de um a N do Enum java.lang.annotation.RetentionPolicy são detalhados (JOY et al., 2000).

[1] É interessante restringir o local onde as anotações podem ser inseridas para evitar uma má interpretação durante sua recuperação via reflexão

Tabela 2 - Parâmetros da anotação retention e seu tempo de duração

Parâmetro	Ciclo de vida
SOURCE	Anotações não são carregadas nos arquivos ".class"
CLASS	Anotações são mantidas nos arquivos ".class" mas não são carregados pela JVM
RUNTIME	Anotações são mantidas pelos arquivos ".class" e carregados pela JVM

Fonte - Adaptado de Joy et al. (2000)

O parâmetro SOURCE é utilizado geralmente por *Integrated Development Environment* (IDE), para adicionar algum comportamento específico, como a anotação @SuppressWarnings que serve para ignorar algum aviso, presente em IDE's como o Eclipse e o IntelliJ IDEA. Para que uma anotação seja recuperada em tempo de execução é preciso adicionar a RetentionPolicy Runtime, devido ao comportamento padrão da anotação @Retention que é o valor CLASS (JOY et al., 2000).

É possível adicionar propriedades em uma anotação para complementar comportamentos ou informações que esta proporcionará, as propriedades permitidas são: Tipos primitivos; Enums; Class (Apenas a classe Class); String; Anotações; e Arrays dos tipos anteriores conforme Figura 11.

Figura 11 - Propriedades permitidas em anotações

```
1  Anotacao anotacoes();
2  Class classes();
3  String strings();
4  boolean boleanos();
5  byte bytes();
6  char chars();
7  double doubles();
8  float floats();
9  int inteiros();
10 long longs();
11 short shorts();
12 Anotacao[] arrayDeAnotacoes();
13 Class[] arrayDeClasses();
14 String[] arrayDeStrings();
15 boolean[] arrayDeBoleanos();
16 byte[] arrayDeBytes();
17 char[] arrayDeChars();
18 double[] arrayDeDoubles();
19 float[] arrayDeFloats();
20 int[] arrayDeInteiros();
21 long[] arrayDeLongs();
22 short[] arrayDeShorts();
```

Fonte - Adaptado de Joy et al. (2000)

Na Figura 12 uma anotação é declarada para exemplificação, esta é mantida até o tempo de execução e pode ser adicionada em outras anotações e métodos. Por padrão, nenhum comportamento é adicionado ou alterado com a adição de uma anotação, isto é feito a partir de métodos que verificam sua existência, como a reflexão, citada no capítulo anterior (BLOCH et al., 2004).

Figura 12 - Declaração de uma anotação

```
1 @Target({ ANNOTATION_TYPE, METHOD })
2 @Retention(RUNTIME)
3 public @interface Anotacao {
4 }
```

Fonte - Produção do autor

Frameworks baseados em reflexão e metadados são *softwares* que utilizam a reflexão para obtenção de metadados, estes disponibilizam anotações com o valor Runtime para que sejam lidos posteriormente (GUERRA; SOUZA; FERNANDES, 2009).

2.2 Framework

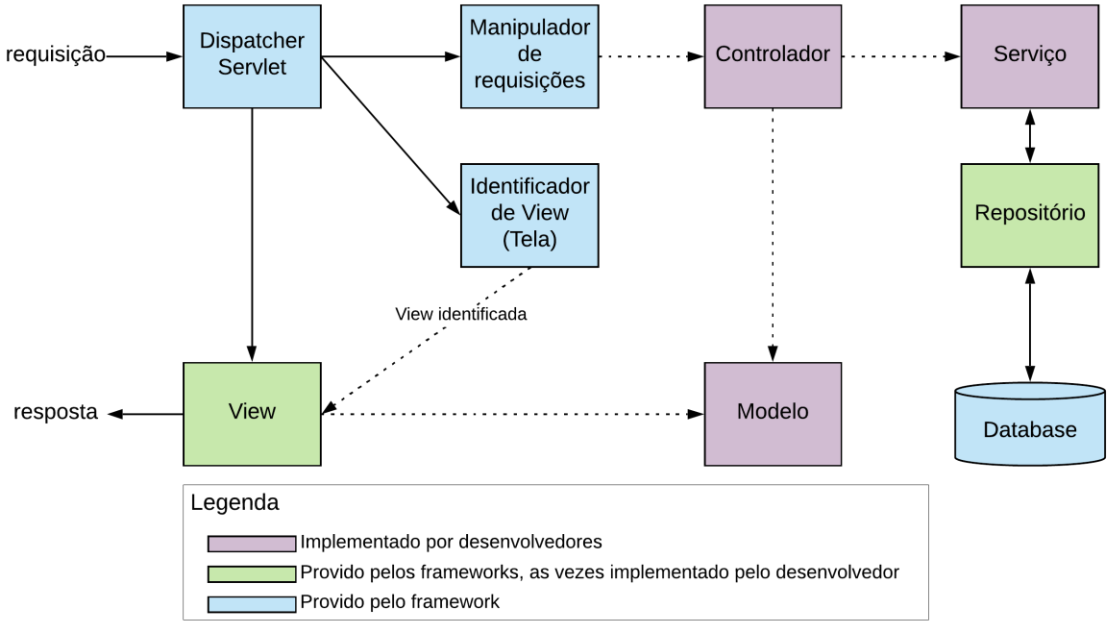
Um *framework* é considerado um *software* incompleto que é especializado com o comportamento de uma aplicação externa (JOHNSON; FOOTE, 1988). Este determina a arquitetura que a aplicação utilizará, sua organização, como: convenções de código, arquitetura do projeto, arquivos externos de configuração e/ou anotações. Isto é definido para que o desenvolvedor tenha que se preocupar apenas com o problema que está resolvendo.

A forma como o *framework* realiza esta organização deve ser baseada no que é mais viável para solucionar a situação comum em relação ao problema encontrado, permitindo que a tarefa repetitiva ou específica seja reaproveitada em novos projetos.

Baseado nisso *frameworks* permitem que aplicações com estruturas semelhantes sejam criadas, facilitando a manutenção, padronização e legibilidade do código, entretanto, isto restringe o desenvolvedor a solução que o *framework* aplica, não permitindo que determinados caminhos sejam seguidos ou certas decisões sejam tomadas no projeto (GAMMA, 2009). A ideia por trás dessa restrição citada é a inversão de controle existente, onde os *frameworks* chamam o

código que foi desenvolvido, não o contrário. Desta forma muitas vezes é realizada a injeção de dependências neste código criado, que é o fornecimento de implementações de interfaces em tempo de execução, a Figura 13 exemplifica uma situação em que existe a inversão de controle em um ambiente *web*, onde a requisição é recebida pelo servidor e a responsabilidade inicial é do *framework*, que realiza o processamento necessário passando a requisição por filtros e outras validações, para depois passar a responsabilidade para as classes criadas pelo desenvolvedor final.

Figura 13 - Inversão de controle



Fonte – Adaptado de Overview… (2013)

Existem pontos extensíveis em *frameworks*, estes são chamados de *hotspots* e servem para especializações como no exemplo anterior (Figura 13) em que os blocos roxos são os *hotspots* fornecidos. Estes são implementados pelos desenvolvedores pois é possível atender requisitos como filtros genéricos que precisam ser verificados e restrições recuperadas de metadados, porém funcionalidades específicas não podem ser atendidas em todos os casos e precisam ser implementadas via *hotspots*.

Geralmente *frameworks* disponibilizam algumas especializações prontas para *hotspots* (blocos verdes da Figura 13), mas caso seja necessário outras podem ser criadas para situações

em que as soluções existentes não resolverem o problema. Por outro lado, ao contrário dos *hotspots* existem os *frozenspots* (blocos azuis da Figura 13) que são trechos que não devem ser alterados ou especializados, são as funcionalidades e processos automatizados pelo *framework* (MARKIEWICZ; LUCENA, 2001).

2.2.1 *Frameworks* reflexivos baseados em metadados

Frameworks reflexivos baseados em metadados revolucionaram a ideia do desenvolvimento destas soluções, devido ao problema de acoplamento existente em *frameworks* tradicionais (GUERRA, 2010). Era possível adicionar comportamentos neste tipo de *framework* via implementação de interfaces e herança de classes abstratas (*hook methods*), o que causa alto acoplamento. Quando o número de dependentes da solução cresce, a manutenção e até mesmo a criação de novos recursos fica complexa, porque alterando o *framework* todos os dependentes também precisarão alterar suas implementações se optarem por utilizar a última versão desenvolvida.

Para solucionar este problema, *frameworks* baseados em reflexão e metadados foram criados, nestes é possível adicionar novos comportamentos com metadados, sendo elas: Convenções de código; Anotações; Arquivos externos; e Definições programáticas.

Convenções de código como *getters* e *setters* dos Java Beans por exemplo, são utilizados por *frameworks* quando possuem algum valor semântico, a Figura 14 exibe como *query methods* são utilizados pelo Spring JPA, nele os parâmetros de consulta são inseridos no nome do método e o Structured Query Language (SQL) é criado em tempo de execução, o Spring identifica que a interface possui *query methods* com a extensão de interfaces parametrizadas em sua documentação, estas são: Repository, CrudRepository e JpaRepository. Convenções de código são limitadas e quando é necessário expressar informações mais complexas, a criação de metadados com esta abordagem não é viável.

Arquivos externos são muito verbosos porque necessitam descrever precisamente onde as modificações acontecerão ou onde estão agregando informações, a Figura 15 exibe um arquivo de configuração básico de persistência de uma entidade para o Hibernate.

Figura 14 - Query method Spring JPA

```
1 public interface CrudRepository<T, Integer> {
2     Optional<T> findById(Integer id);
3 }
```

Fonte – Gierke et al. (2019)

Figura 15 - Configuração de persistência de uma classe via XML

```
1  <?xml version="1.0"?>
2  <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
3  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
4
5  <hibernate-mapping>
6     <class name="br.fatec.sjc.exemplo.Pessoa" table="pessoa" catalog="exemplo">
7        <id name="pessoaId" type="java.lang.Integer">
8           <column name="PES_ID" />
9           <generator class="identity" />
10       </id>
11       <property name="nome" type="string">
12          <column name="PES_NOME" length="50" not-null="true" unique="false" />
13       </property>
14       <property name="cpf" type="string">
15          <column name="PES_CPF" length="11" not-null="true" unique="true" />
16       </property>
17    </class>
18 </hibernate-mapping>
```

Fonte - Adaptado de Jboss (b)

Para resolver estes problemas as anotações foram criadas conforme JCP 269 (DARCY, 2005). Com as anotações os metadados são inseridos no próprio código, deixando a semântica mais fluída e menos verbosa. Na Figura 16 é realizada a mesma configuração da Figura 15, neste exemplo é possível analisar a facilidade que as anotações trouxeram para o desenvolvimento, possibilitando que metadados sejam adicionados de uma forma mais natural, fazendo parte da própria linguagem.

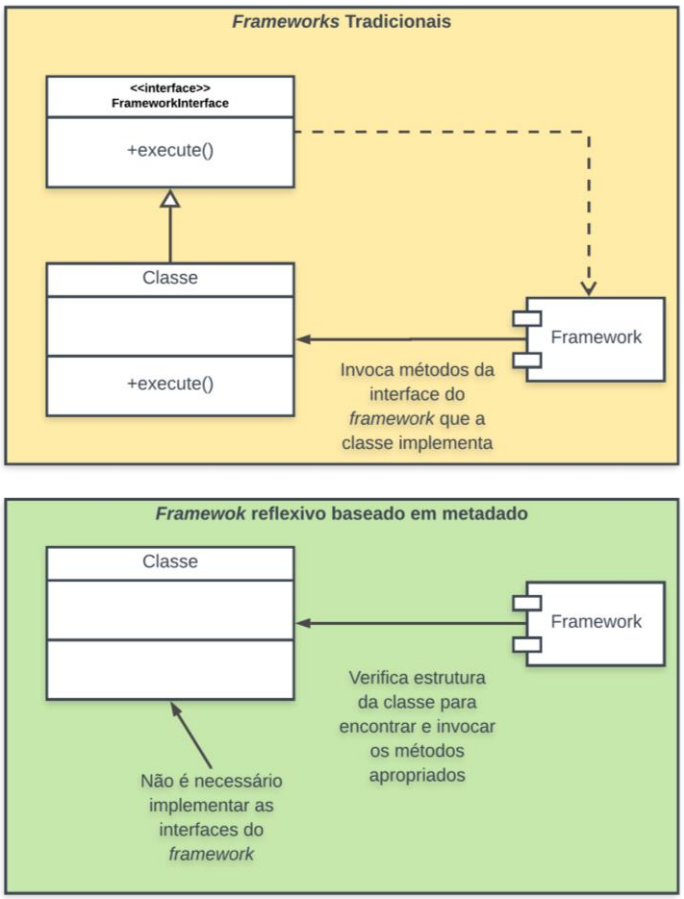
Estas informações são lidas utilizando técnicas de introspecção citadas anteriormente, desta forma o acoplamento existente é baseado nos metadados, assim o *framework* e seus dependentes podem ser modificados sem grandes problemas. A Figura 17 exibe a diferença entre o acoplamento existente em *frameworks* tradicionais e baseados em reflexão e metadados.

Figura 16 - Configuração de persistência de uma classe via anotação

```
1  @Entity
2  @Table(name = "pessoa", uniqueConstraints = {
       @UniqueConstraint(columnNames = { "cpf" })
3  })
4  public class Pessoa implements Serializable {
5
6      @Id
7      private Integer id;
8      @Column(nullable = false)
9      private String nome;
10     @Column(nullable = false)
11     private String cpf;
12
13     // Getters e setters omitidos
14
15 }
```

Fonte – Adaptado de Jboss (a)

Figura 17 - Frameworks tradicionais e reflexivos



Fonte - Adaptado de Silva, Guerra e Fernandes (2013b)

Com esta Figura é possível identificar a diferença entre as arquiteturas dos *frameworks*. Os reflexivos utilizam técnicas de PD para realizar a interceptação e a verificação da estrutura das classes a procura de metadados.

2.3 Gamificação

Gamificação pode ser definida como a utilização de conceitos de *design* de jogos, como: Ganho de pontos, *ranking*, troféus, e *rewards* em aplicações de outros contextos (DETERDING et al., 2011). *Softwares* de *e-commerce*, *sites* de *e-learning*, são alguns exemplos de contextos onde a gamificação pode ser aplicada. Um exemplo de aplicação que utiliza gamificação é a Foldit (BURKE, 2012), que utiliza conceitos de pontos e troféus em um jogo de quebra cabeça, para na verdade, prever a estrutura da proteína humana (DETERDING et al., 2011). O propósito da gamificação é fazer com que a utilização das aplicações não diminua com o tempo e proporcione experiências melhores ao usuário, pois as conquistas adquiridas durante a utilização incentivam a continuidade. A Psicologia Comportamental trata desta abordagem utilizada na gamificação como reforços positivos e negativos (SKINNER, 1990). Os reforços positivos têm como objetivo estimular indivíduos a repetirem ações a partir da agregação de algo, na gamificação isto é aplicado com a ideia da aquisição das conquistas, e como já dito, isto estimula o indivíduo a continuar os processos que está fazendo para receber mais conquistas. Por outro lado, existem os reforços negativos, que incentivam o indivíduo a realizar o processo inverso do que foi feito para receber este estímulo, isto é, o comportamento é condicionado a partir de processos para evitar alguma situação, como por exemplo perder pontos ou perder posições no ranking se tratando de gamificação, isto gera efeitos colaterais como fuga e esquiva no indivíduo e este aprende naturalmente a evitar reforços negativos (LINEHAN; KIRMAN; ROCHE, 2015). Nas duas formas é possível condicionar o indivíduo a realizar atividades para chegar a um propósito.

2.4 Esfinge Project

Esfinge Project[2] é um projeto *open-source* iniciado em 2011 por Dr. Eduardo Guerra junto a empresa GSW, que tem como objetivo a criação de soluções reutilizáveis para um desenvolvimento ágil, um produto final flexível e de fácil manutenção.

O projeto disponibiliza 9 *frameworks* até o momento, estes são: QueryBuilder, Comparison, Guardian, AOM Role Mapper, SystemGlue, Gamification, Metadata, Classmock, ReTest.

Todos os *frameworks* citados acima seguem uma filosofia que consiste em: Configuração de metadados para que o comportamento desejado ocorra; Componentes que podem ser integrados a aplicações de forma simples; Pontos de extensão para criação de novas funcionalidades; Remover a preocupação com a solução que o *framework* disponibiliza, permitindo que o desenvolvedor foque apenas em sua aplicação específica (GUERRA et al., 2011b).

2.4.1 Esfinge Gamification

O Esfinge Gamification[3] é um *framework* que aplica mecanismos de gamificação para *softwares* que necessitam destes processos. Independente do domínio do *software* é possível utilizar o *framework*, pois este é desacoplado de lógicas da aplicação, sua responsabilidade é a tratativa dos dados de gamificação, portanto, pode ser integrado a qualquer programa Java, permitindo que o desenvolvedor foque na solução que está trabalhando, e deixe as responsabilidades de gamificação para o framework (GUERRA et al., 2011a).

O comportamento do *framework* é especificado via metadados conforme a filosofia dos projetos Esfinge citada anteriormente, estes são anotações que podem ser adicionadas em programas Java, a Tabela 3 detalha as opções disponíveis. Existem quatro tipos de processos implementados pelo *framework*, estes são: Ponto, Ranking, *Reward* e Troféu, vide Figura 18.

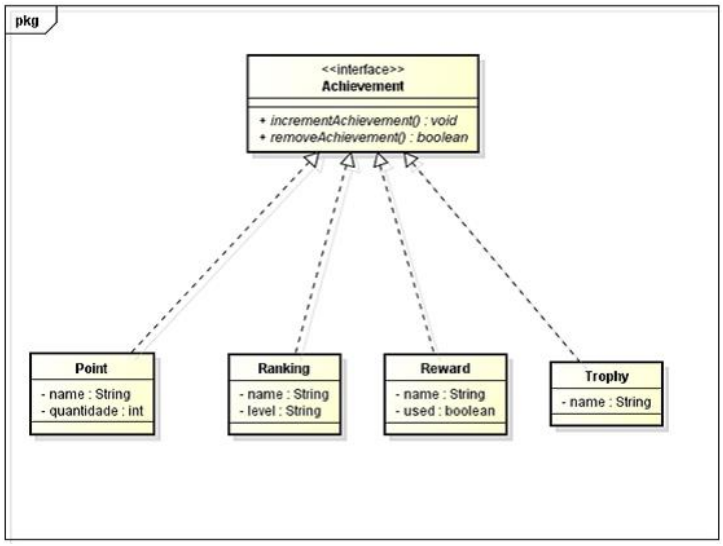
[2] O projeto está disponível no endereço: http://esfinge.sf.net.
[3] A documentação do projeto está disponível no endereço: http://esfinge.sourceforge.net/Gamification.html.

Tabela 3 - Anotações disponibilizadas pelo Esfinge Gamification

Anotação	Comportamento
@PointsToUser, @RankingsToUser, @RewardsToUser, @TrophiesToUses	Incremento de conquista para o usuário atual.
@RemoveRankings, @RemovePoints, @RemoveReward, @RemoveTrophy	Eliminação de conquista do usuário atual.
@PointToParam, @RankingToParam, @RewardToParam, @TrophyToParam	Incremento de conquistas para outro usuário, ou seja, a ação do usuário logado no sistema gera pontos para outro usuário.
@RemovePointsToParam	Remove Pontos de outro usuário seguindo a lógica de atribuição, isto é, o usuário que está logado no sistema desencadeia este comportamento.
@TrophyWhenReachPointLimit	Cria um EventListener para adicionar troféus quando uma quantidade de Pontos for alcançada.

Fonte – Adaptado de Guerra et al. (2011a)

Figura 18 - Conquistas implementadas no projeto Esfinge Gamification



Fonte – Guerra et al. (2011a)

Caso as implementações existentes não atendam completamente as funcionalidades necessárias em alguma situação, é possível implementar a interface Achievement e criar metadados para a nova implementação.

2.4.1.1 Características das conquistas disponibilizadas

Ponto: Tem como intenção atribuir determinada quantidade de pontos e seu tipo, por exemplo, uma conquista que atribui 10 pontos de moedas de ouro, onde moedas de ouro são o tipo e 10 os pontos atribuídos.

Ranking: Se assemelha a uma hierarquia militar, onde o ranking é de status, não de posições. Por exemplo, um usuário possui o status iniciante quando começa a utilizar a aplicação, e quando realiza algum processo específico, é premiado com o status de intermediário ou avançado.

Troféu: Pode ser atribuído apenas uma vez, por exemplo, caso um usuário realize um processo e receba um troféu por isto, na próxima vez que realizar o mesmo processo não receberá outro troféu.

Reward: É uma conquista que é consumida, ou seja, fica indisponível após ser usada. Por exemplo, um *reward* de bônus de ligações será recebido não consumido por padrão, isto é, disponível para ser usado. Este só será consumido quando uma ligação for realizada com seu recurso de bônus, e após isso ficará indisponível.

2.4.1.2 Pontos de extensão

O *framework* possui dois pontos de extensão (*hotspots*), onde novos comportamentos podem ser adicionados quando necessário. Desta forma é possível aplicar outras lógicas de gamificação, implementando a interface Achievement (Figura 18) disponível no pacote net.sf.esfinge.gamification.achievement, caso os comportamentos disponibilizados pelo *framework* não sejam adequados à determinada situação.

A interface Achievement exibida na Figura 19 possui três métodos, eles servem para realizar os respectivos processos em Achievements: Recuperação de nome; Incremento de informações (adição de pontos por exemplo); e remoção de informações.

Figura 19 - Interface Achievement

```
1  public interface Achievement {
2
3      public String getName();
4
5      public void incrementAchievement(Achievement a);
6
7      public boolean removeAchievement(Achievement r);
8  }
```

Fonte – Adaptado de Guerra et al. (2011a)

Implementando esta interface novas ações podem ser adicionados via anotações, elas possuirão a anotação @GamificationProcessor exibida na Figura 20, que recebe uma classe que implemente a interface AchievementProcessor como parâmetro.

Figura 20 - Anotação @GamificationProcessor

Fonte - Adaptado de Guerra et al. (2011a)

```
1  // anotacoes retention e target omitidas
2  public @interface GamificationProcessor {
3      Class<? extends AchievementProcessor> value();
4  }
```

A Figura 21 exemplifica a anotação @PointsToUser, que possui a classe PointsToUserProcessor (Figura 22) como implementação de AchievementProcessor. Cada anotação criada possuirá uma implementação de AchievementProcessor que executa o comportamento específico da anotação, isto é, a anotação adiciona a informação e o AchievementProcessor seu comportamento.

Figura 21 - Anotação @PointsToUser

```
1  // anotacoes target e retention omitidas
2  @GamificationProcessor(PointsToUserProcessor.class)
3  public @interface PointsToUser {
4      int quantity();
5      String name();
6  }
```

Fonte – Adaptado de Guerra et al. (2011a)

Figura 22 - Implementação de AchievementProcessor que define comportamento da anotação @PointsToUser

```
1  public class PointsToUserProcessor implements AchievementProcessor {
2
3      private int quantity;
4      private String name;
5
6      @Override
7      public void receiveAnnotation(Annotation an) {
8          PointsToUser ptu = (PointsToUser) an;
9          quantity = ptu.quantity();
10         name = ptu.name();
11     }
12
13     @Override
14     public void process(Game game, Object encapsulated, Method method,
           Object[] args) {
15         Object user = UserStorage.getUserID();
16         Point p = new Point(quantity, name);
17         game.addAchievement(user, p);
18     }
19 }
```

Fonte - Adaptado de Guerra et al. (2011a)

A interface AchievementProcessor exibida na Figura 23 possui dois métodos a serem implementados, estes são: receiveAnnotation() e process(). Eles têm o objetivo de recuperar informações da anotação recebida e executar seu comportamento, respectivamente. A Figura 22 exibe um exemplo onde as informações quantity e name são recuperadas da anotação @PointsToUser entre as linhas 8 e 10. O comportamento desta implementação é a atribuição de pontos, portanto o método addAchievement() é invocado para que isso seja persistido de acordo com a especialização da classe Game.

Figura 23 - Interface AchievementProcessor

```
1  public interface AchievementProcessor {
2
3      public void receiveAnnotation(Annotation an);
4
5      public void process(Game game, Object encapsulated, Method method,
           Object[] args);
6
7  }
```

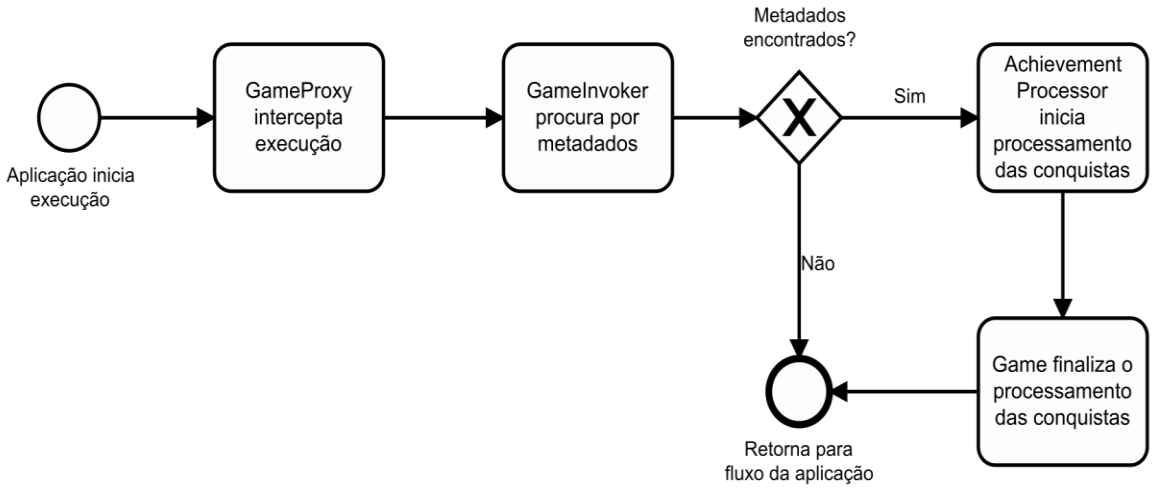
Fonte - Adaptado de Guerra et al. (2011a)

Também é possível adicionar novos comportamentos de persistência estendendo a classe abstrata Game, disponível no pacote net.sf.esfinge.gamification.mechanics. Atualmente existem três implementações, estas são: Persistência em arquivo de propriedades; Banco de dados relacional; e em memória (GUERRA et al., 2011a).

2.4.1.3 Funcionamento do Esfinge Gamification

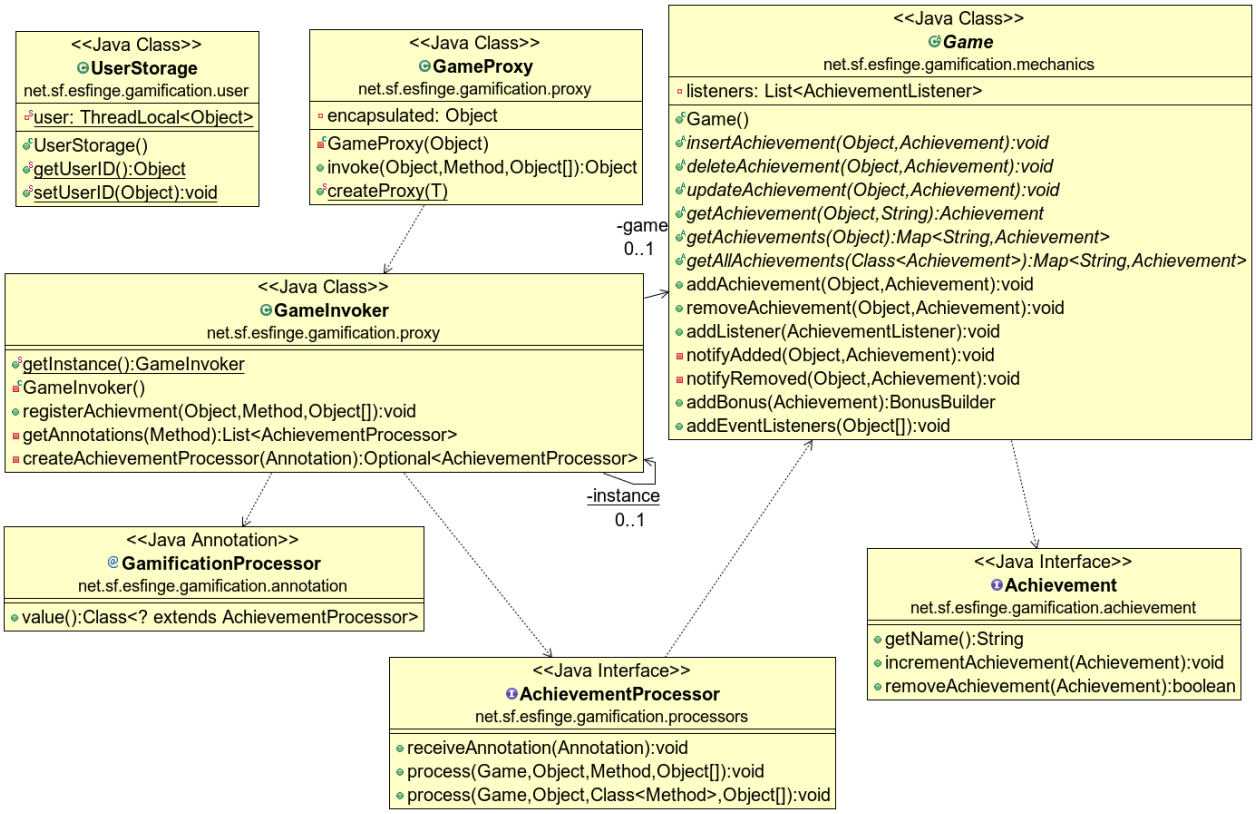
A Figura 24 apresenta o diagrama de classe do projeto, já o diagrama apresentado na Figura 25 exibe o fluxo de funcionamento do Esfinge Gamification. A classe UserStorage possui dois métodos, setUserID() e getUserID(), responsáveis por definir o usuário que será utilizado nos processos de gamificação e recuperar este usuário, respectivamente.

Figura 24 - Fluxo de funcionamento do Esfinge Gamification



Fonte - Produção do autor

Figura 25 - Diagrama de classes do projeto Esfinge Gamification



Fonte – Produção do autor

O método estático createProxy() da classe GameProxy é responsável pela criação de um PD que inicia a execução do Esfinge Gamification. É possível analisar na Figura 26 que a estrutura utilizada pelo *framework* é semelhante a Figura 8. Os diferenciais são as validações realizadas entre as linhas 18 e 23, pois, se durante a criação alguma inconsistência nos metadados for encontrada, como as anotações @PointsToUser e @RemovePoints no mesmo local[4], a exceção GamificationConfigurationException do pacote net.sf.esfinge.gamification.exception é lançada.

[4] @PointsToUser atribui pontos e @RemovePoints remove pontos vide Tabela \ref{tab:anotacoes-gamification}

Figura 26 - Criação de *proxy* dinâmico do Esfinge Gamification

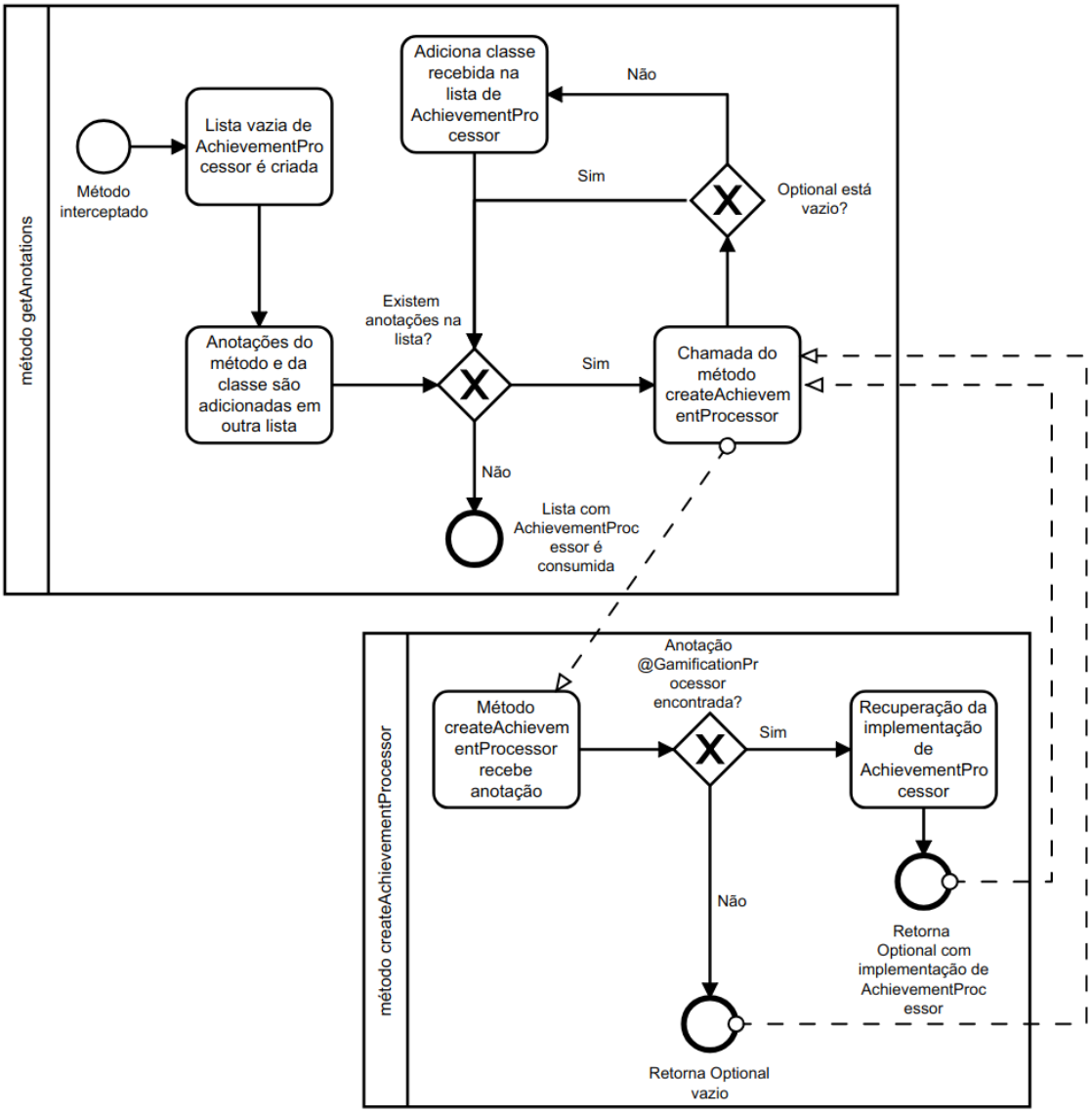
```
1  public class GameProxy implements InvocationHandler {
2          private Object encapsulated;
3
4      private GameProxy(Object encapsulated) {
5          this.encapsulated = encapsulated;
6          }
7
8  //implementacao do metodo invoke omitida
9
10     public static <T> T createProxy(T encapsulated) {
11         Object obj = Proxy.newProxyInstance(
12                 encapsulated.getClass().getClassLoader(),
13             encapsulated.getClass().getInterfaces(),
14             new GameProxy(encapsulated)
15         );
16
17         try {
18             MetadataValidator.validateMetadataOn(
19                 encapsulated.getClass()
20             );
21             for (Class interf :
                   encapsulated.getClass().getInterfaces()) {
22                 MetadataValidator
23                 .validateMetadataOn(interf);
24             }
25         } catch (AnnotationValidationException e) {
26             throw new GamificationConfigurationException("Invalid
                   annotation configuration", e);
27         }
28
29         return (T) obj;
30     }
31 }
```

Fonte - Adaptado de Guerra et al. (2011a)

A Figura 27 detalha os processos macro da Figura 24. Quando o PD intercepta execuções, este procura por metadados na classe e no método que foi interceptado, como é possível identificar no segundo passo do processo "método getAnnotations". Cada anotação encontrada é passada para o processo "método createAchievementProcessor" que procura pelos metadados citados na Tabela 3 retornando um Optional com a implementação de AchievementProcessor caso o metadado possua a anotação @GamificationProcessor citada anteriormente, para que esta

implementação seja adicionada a uma lista de AchievementProcessors criada, a fim de executar os métodos process(), aplicando o comportamento específico das anotações.

Figura 27 - Processo de recuperação de metadados



Fonte - Produção do autor

2.4.2 Esfinge Guardian

Autorização em *software* pode ser resumida na pergunta: "Esta entidade pode acessar este recurso?" (SANDHU; SAMARATI, 1994). Onde entidade pode ser definida como um usuário ou

sistema externo com a intenção de realizar um processo em um ambiente controlado, e recurso como algo que está sob proteção, algo não público (BARTSCH, 2011).

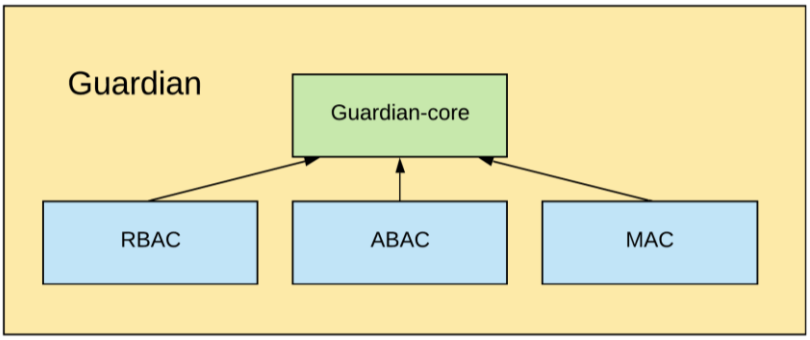
O Esfinge Guardian é um *framework* para a aplicação de autorizações em *softwares*, seguindo a filosofia do Esfinge Project citada anteriormente. Este aplica os seguintes modelos de controle de acesso:

- *Attribute Based Access Control* (ABAC): É um controle de acesso lógico que verifica atributos de objetos, em um determinado ambiente e assunto, para validar se o objeto poderá ou não realizar o acesso naquela situação baseado nas políticas definidas (HU; KUHN; FERRAIOLO, 2015).
- *Mandatory access control* (MAC): É um controle de acesso definido por apenas uma entidade no sistema, e apenas esta entidade pode realizar a alteração das permissões definidas, geralmente este tipo de controle de acesso é utilizado para informações sensíveis, semelhante a um sistema militar (LINDQVIST, 2006).
- *Role Based Access Control* (RBAC): É uma forma de controle de acessos baseado em funções, isto é, cada objeto (este podendo ser um usuário ou um ativo), possui uma ou mais funções que o permitem realizar determinados processos de acordo com seu nível hierárquico, semelhante a uma empresa (SANDHU et al., 2000).

2.4.2.1 Arquitetura

O *framework* é dividido por módulos, estes possuem a arquitetura exibida na Figura 28. Abaixo seus módulos serão explicados:

Figura 28 - Arquitetura dos módulos existêntes no Esfinge Guardian



Fonte - Adaptado de Guerra et al. (2017)

- Guardian: Este é o Super Project Object Model (POM) Foundation (2019) do projeto, tem como objetivo controlar as dependências compartilhadas e o processo de empacotamento dos demais módulos;
- Guardian-core: Tem as funcionalidades essenciais para um modelo de autorização, este modulo é utilizado por todos os modelos de controle de acesso presentes no Esfinge Guardian;
- RBAC, ABAC e MAC: Implementam os modelos de controle de acesso com seus nomes.

2.4.2.2 Pontos de extensão

Nesta seção os *hotspots* do Esfinge Guardian serão abordados. Para estender o *framework* é necessário implementar as interfaces Populator e Authorizer, além da criação de anotações de segurança.

A interface Populator exibida na Figura 29 possui o método populate, que é dedicado a recuperação de informações de segurança necessárias para a autorização.

Figura 29 - Interface Populator

```
1 public interface Populator {
2     void populate(AuthorizationContext context);
3 }
```

Fonte - Adaptado de Silva, Guerra e Fernandes (2013a)

Além de implementar a interface Populator é preciso configurá-la. Atualmente existem duas formas de realizar esta configuração, com a anotação @PopulatorClass ou com o arquivo de configuração chamado "org.esfinge.guardian.populator.Populator". A anotação @PopulatorClass exibida na Figura 30 possui uma propriedade onde o Populator implementado é definido para ser recuperado pelo *framework* (linha 3).

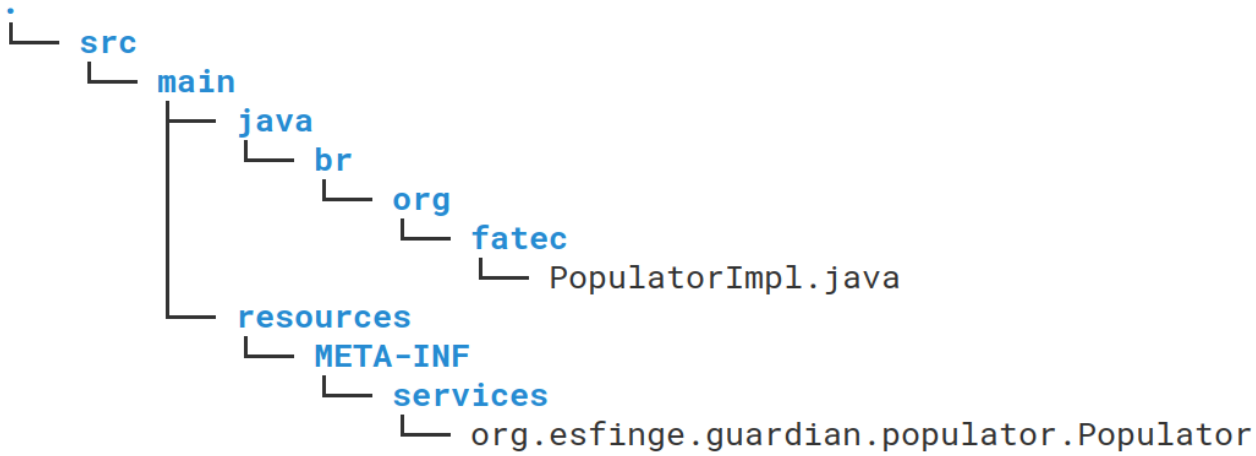
Figura 30 - Anotação PopulatorClass

```
1 //anotacoes retention e target omitidas
2 public @interface PopulatorClass {
3     Class<? extends Populator> value();
4 }
```

Fonte - Adaptado de Silva, Guerra e Fernandes (2013a)

O arquivo de configuração "org.esfinge.guardian.populator.Populator" precisa estar localizado no caminho resources/META-INF/services/ para ser identificado pelo *framework*. O conteúdo deste arquivo é a classe e o pacote que implementam a interface Populator, portanto, se a classe PopulatorImpl localizada no pacote br.org.fatec for configurada desta forma, o conteúdo do arquivo será br.org.fatec.PopulatorImpl e a estrutura de diretórios ficará com o diretório resources localizado no mesmo nível dos pacotes de arquivos fonte, conforme a Figura 31 exemplifica.

Figura 31 - Estrutura de diretórios para identificação da implementação da interface Populator



Fonte - Produção do autor

A interface Authorizer (Figura 32) possui o método authorize, seu retorno é um booleano, que consiste na liberação da autorização ou não. Esta também possui a declaração de um atributo genérico (E extends Annotation), neste atributo é esperado a anotação de segurança que será avaliada pela implementação.

Figura 32 - Interface Authorizer

```
1 public interface Authorizer<E extends Annotation> {
2     Boolean authorize(AuthorizationContext context, E
      securityAnnotation);
3 }
```

Fonte – Adaptado de Silva, Guerra e Fernandes (2013a)

As anotações de segurança precisam conter a anotação @AuthorizerClass (Figura 33), que recebe uma classe que estenda Authorizer como parâmetro.

Figura 33 - Anotação @AuthorizerClass

```
1 //anotacoes retention e target omitidas
2 public @interface AuthorizerClass {
3     Class<? extends Authorizer<? extends Annotation>> value();
4 }
```

Fonte – Adaptado de Silva, Guerra e Fernandes (2013a)

A Figura 34 exemplifica como novas anotações de segurança devem ser implementadas.

Figura 34 - Exemplo de implementação da anotação AuthorizerClass

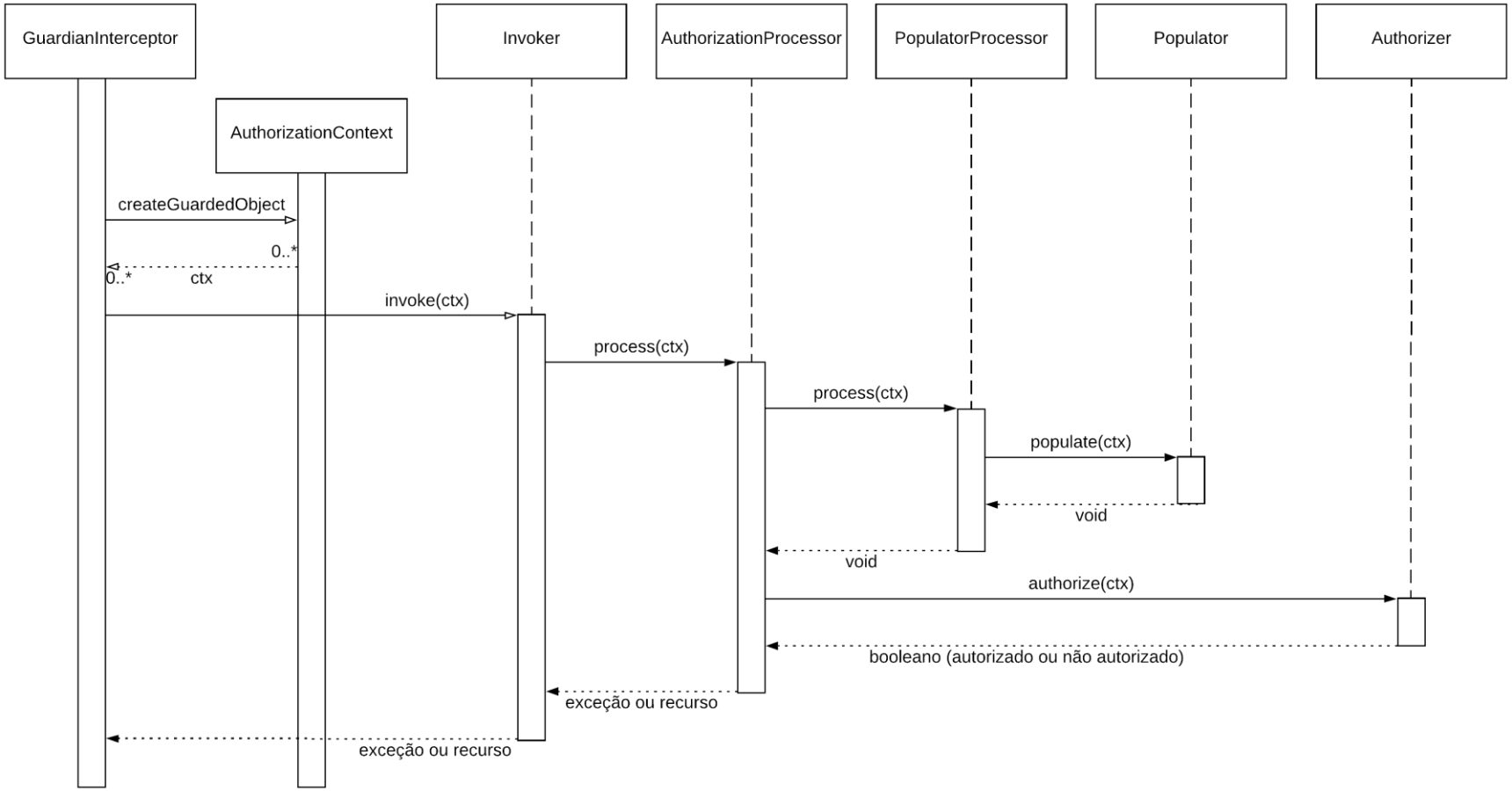
```
1 @Target({ ElementType.METHOD, ElementType.ANNOTATION_TYPE,
     ElementType.TYPE })
2 @Retention(RetentionPolicy.RUNTIME)
3 @AuthorizerClass(AnotacaoSegurancaAuthorizer.class)
4 public @interface AnotacaoSeguranca {
5 }
```

Fonte - Produção do autor

2.4.2.3 Funcionamento do Esfinge Guardian

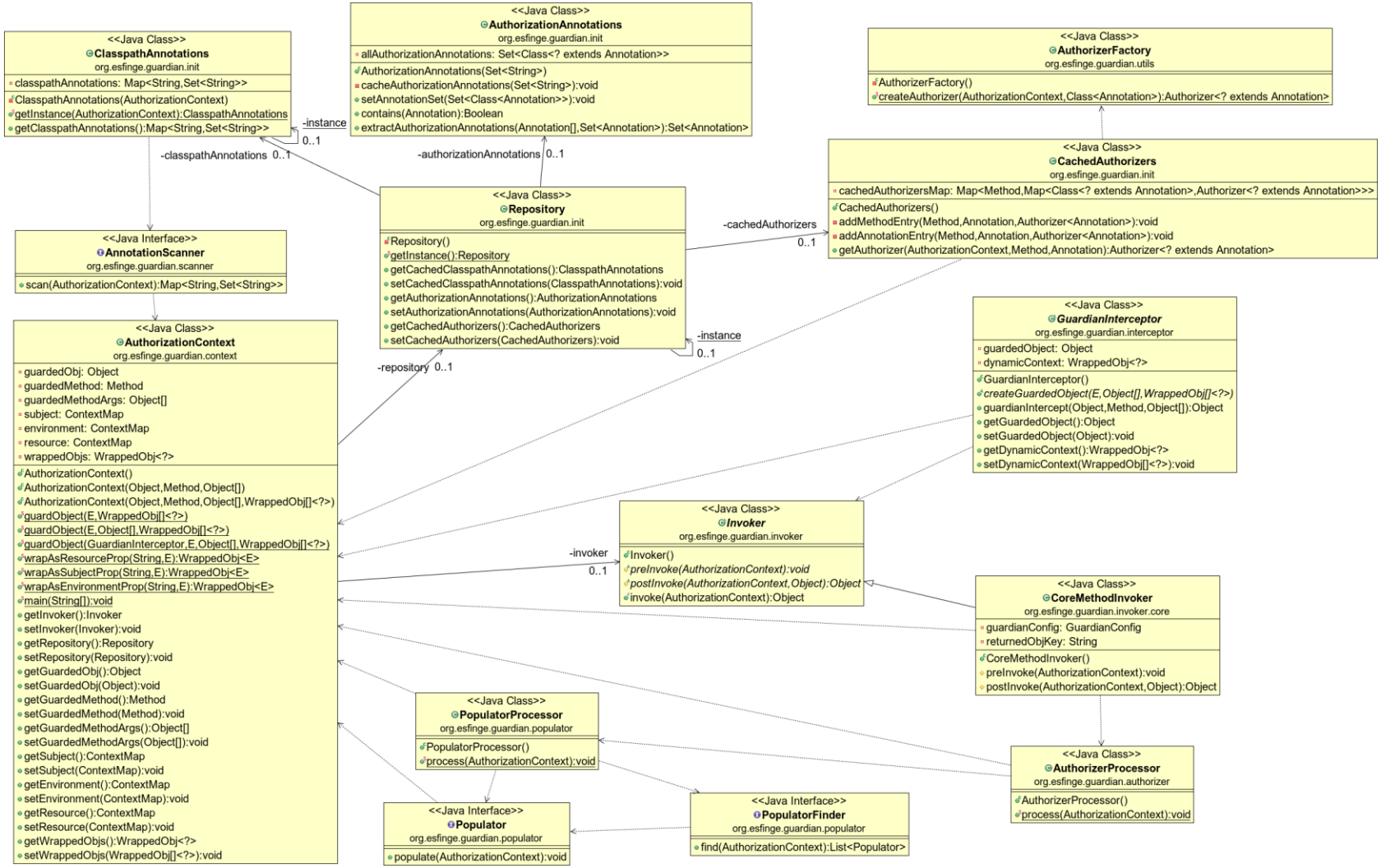
O funcionamento do *framework* pode ser visualizado no diagrama exibido na Figura 35, que tem como objetivo esclarecer o fluxo do diagrama de classes da Figura 36. O diagrama de sequência (Figura 35) tem como início a interceptação de métodos. Quando o *framework* é executado, um contexto de autorização nomeado AuthorizationContext é criado, este contexto contém informações sobre o objeto interceptado vide Figura 36, além de informações que são utilizadas posteriormente para que a autorização ocorra. Após a criação do contexto de autorização, ele é invocado e o processamento da autorização é iniciado (execução de AuthorizationProcessor). Para que este processamento aconteça, primeiro é preciso encontrar as implementações da interface Populator, pois elas possuem as informações necessárias para realização da autorização conforme já dito. Com as implementações encontradas, as autorizações podem ser verificadas a partir da chamada dos Authorizers recuperados dos metadados definidos no método interceptado. Estes realizam as verificações necessárias, e caso alguma seja falsa ou falhe, a exceção AuthorizationException do pacote org.esfinge.guardian.exception é lançada, retornando a mensagem de erro "Unauthorized Access", ou caso nenhuma inconsistência ou erro seja encontrado, é permitido que o método acesse o recurso desejado inicialmente.

Figura 35 - Diagrama de funcionamento do projeto Esfinge Guardian



Fonte - Adaptado de Silva, Guerra e Fernandes (2013b)

Figura 36 - Diagrama de classes do projeto Esfinge Guardian



Produção do autor

3 DESENVOLVIMENTO

Este capítulo detalha as soluções desenvolvidas para o problema citado na introdução.

3.1 Requisitos

Nesta seção os requisitos funcionais e não funcionais serão descritos.

3.1.1 Requisitos Funcionais

RF01 - Prover autorização com os dados de gamificação;

RF02 - Acionar o Esfinge Guardian apenas quando existirem anotações de segurança;

RF03 - Registrar *logs* das validações de segurança realizadas.

3.1.2 Requisitos não funcionais

RNF01 - Baixo acoplamento entre o *framework* e aplicações;

RNF02 - Pontos de extensibilidade (*hotspots*) para criação de novas formas de autorização;

RNF03 - O fluxo de funcionamento do Esfinge Gamification não deve ser alterado pelo Esfinge Guardian quando não existirem anotações de segurança;

RNF04 - Os *frameworks* Esfinge Gamification e Esfinge Guardian devem ser integrados;

RNF05 - A autorização deve ser realizada pelo *framework* Esfinge Guardian.

3.2 Visão geral

Esta seção tem como objetivo exemplificar como o *framework* deve ser usado. No exemplo escolhido para explicação um ambiente foi criado, em que notas podem ser adicionadas apenas por usuários que possuem o Ranking Avaliator e o Level Master. A Figura 37 apresenta a configuração do Esfinge Gamification neste ambiente. Na linha 10 a forma de persistência é especificada; a linha 13 é responsável por definir a instância de Game utilizada pelo *framework*; o usuário utilizado nas lógicas de validação e gamificação é definido na linha 14; já na linha 17 um PD é criado para ser interceptado pelo Esfinge Gamification.

Figura 37 - Configuração e utilização do *framework*

```
1  public class AuthorizationSample {
2
3      public static void main(String[] args) {
4
5          User user = new User();
6          user.setClassroom("C12019");
7          user.setRa("C1A252019");
8
9          // Metodo de persistencia de conquistas escolhido foi o
               armazenamento de memoria
10         Game game = new GameMemoryStorage();
11
12         // Configuracao do Esfinge Gamification
13         GameInvoker.getInstance().setGame(game);
14         UserStorage.setUserID(user.getRa());
15
16         // Cria um PD para ser interceptado
17         Person pdUser = GameProxy.createProxy(user);
18
19         pdUser.addNote(10.0);
20
21     }
22 }
```

Fonte - Produção do autor

Neste exemplo o usuário definido para ser utilizado nas lógicas de validação e gamificação do *framework* tenta alterar suas notas na linha 19 e recebe a exceção "Unauthorized

Access" do pacote org.esfinge.guardian.exception.AuthorizationException devido a falta do Ranking e do Level necessários definidos anteriormente conforme Figura 38.

Figura 38 - Exceção Unauthorized Access durante execução de método sem as autorizações definidas

```
@net.sf.esfinge.metadata.annotation.validator.NotNull()
@net.sf.esfinge.metadata.annotation.validator.NotNull()
@net.sf.esfinge.metadata.annotation.validator.NotNull()
@net.sf.esfinge.metadata.annotation.validator.NotNull()
mai 13, 2019 6:09:17 PM net.sf.esfinge.gamification.guardian.auth.ranking.AllowRankingAndLevelAuthorizer authorize
WARNING: Unauthorized accesss: Denied achievement: Ranking Avaliator and Level Master
Exception in thread "main" org.esfinge.guardian.exception.AuthorizationException: Unauthorized Access
```

Fonte - Produção do autor

A Figura 39 exibe a configuração realizada na classe User para que a autorização seja monitorada pelo Esfinge Guardian. A linha 11 exibe a anotação @AllowRankingAndLevel que define a regra citada no início desta seção.

Figura 39 - Configuração de segurança da classe User

```
1  public class User implements Person {
2
3      private String classroom;
4      private String ra;
5      private List<Double> notes;
6
7  // constructor omitido
8
9  // Anotacao de configuracao
10         @Override
11     @AllowRankingAndLevel(achievementName = "Avaliator", level =
           "Master")
12     public void addNote(Double note) {
13         List<Double> notes = this.getNotes();
14         Objects.requireNonNull(notes, "Notes can't be null");
15         notes.add(note);
16     }
17
18 // Getters and setters omitidos
19 }
```

Fonte - Produção do autor

Para que notas pudessem ser adicionadas pelo usuário seria necessário adicionar o Ranking utilizado na regra de autorização, descrita no início desta seção. O exemplo apresentado

na Figura 40 exibe um novo método criado para a atribuição desta autorização por meio da adição do Achievement necessário.

Figura 40 - Interface Person

```
1  public interface Person {
2
3      void addNote(Double note);
4
5      @RankingsToUser(name = "Avaliator", level = "Master")
6      void promotePerson();
7  }
```

Fonte - Produção do autor

A única alteração feita para a adição das notas ser autorizada é a ordem das chamadas, como é possível observar na Figura 41 onde o método promotePerson() é invocado na linha 16, para que o Achievement utilizado na autorização seja adicionado, após isto a adição da nota é feita pelo usuário na linha 17.

A Figura 42 apresenta a execução autorizada pelo *framework*, nesta é possível verificar que a mensagem exibida em vermelho é "Authorized access".

Figura 41 - Operação autorizada pelo Esfinge Guardian

```
1  public class AuthorizationSample {
2
3      public static void main(String[] args) {
4
5          User user = new User();
6          user.setClassroom("C12019");
7          user.setRa("C1A252019");
8
9          Game game = new GameMemoryStorage();
10         GameInvoker.getInstance().setGame(game);
11         UserStorage.setUserID(user.getRa());
12
13         Person pdUser = GameProxy.createProxy(user);
14
15         // a anotacao neste metodo faz o usuario possuir o Achievement
               utilizado na regra de autorizacao
16         pdUser.promotePerson();
17         pdUser.addNote(10.0);
18     }
19 }
```

Fonte - Produção do autor

Nas Figuras 38 e 42 os logs citados no fluxo de funcionamento do *framework* (Figura 24) podem ser visualizados. Estes têm como objetivo evidenciar a análise do Esfinge Guardian durante a interceptação.

Figura 42 - Execução autorizada pelo *framework*

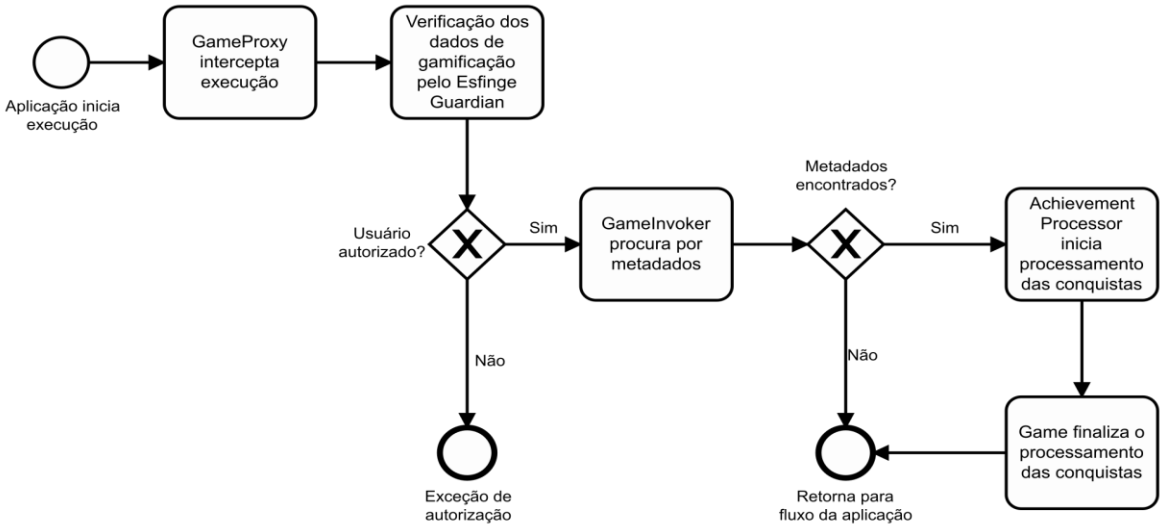
```
@net.sf.esfinge.metadata.annotation.validator.NotNull()
@net.sf.esfinge.metadata.annotation.validator.NotNull()
@net.sf.esfinge.metadata.annotation.validator.NotNull()
@net.sf.esfinge.metadata.annotation.validator.NotNull()
Congratulations user C1A252019 you have won the ranking Avaliator with Master level
mai 13, 2019 7:01:13 PM net.sf.esfinge.gamification.guardian.auth.ranking.AllowRankingAndLevelAuthorizer authorize
INFO: Authorized accesss: Required achievement: Ranking Avaliator and Level Master
```

Fonte - Produção do autor

3.3 Fluxo de funcionamento

Para que os novos recursos fossem adicionados ao Esfinge Gamification, foi necessário alterar seu fluxo de funcionamento exibido anteriormente na Figura 24 a fim de adicionar os processos de autorização. Após as modificações realizadas o *framework* agora possui o fluxo de funcionamento da Figura 43.

Figura 43 - Fluxo de funcionamento atual do Esfinge Gamification



Fonte - Produção do autor

Neste fluxo, aplicações em que o Esfinge Gamification foi configurado, são interceptadas via PD conforme já dito, porém o diferencial é que antes do fluxo de gamificação ser executado o Esfinge Guardian é invocado para que validações de autorização do usuário sejam realizadas.

Na linha 9 da Figura 44 um objeto que será interceptado pelo Esfinge Guardian é criado, e na linha 15 em que o método interceptado pelo PD do Esfinge Gamification é invocado, o objeto passado como parâmetro para a execução é o PD do Esfinge Guardian.

Figura 44 - Criação do *proxy* dinâmico do Esfinge Gamification

```
1  public class GameProxy implements InvocationHandler {
2
3      private Object encapsulated;
4      private Object guardedObject;
5
6      private GameProxy(Object encapsulated) {
7          this.encapsulated = encapsulated;
8          // tratativa de excecao omitida
9          this.guardedObject =
               AuthorizationContext.guardObject(encapsulated);
10
11     }
12
13     public Object invoke(Object proxy, Method method, Object[] args)
           throws Throwable {
14         try {
15             Object returnValue = method.invoke(guardedObject, args);
16             GameInvoker gameInvoker = GameInvoker.getInstance();
17             gameInvoker.registerAchievment(encapsulated, method, args);
18
19             return returnValue;
20         } catch (InvocationTargetException e) {
21             throw e.getTargetException();
22         }
23     }
24     //metodo de criacao do proxy dinamico ja exibido foi omitido
25 }
```

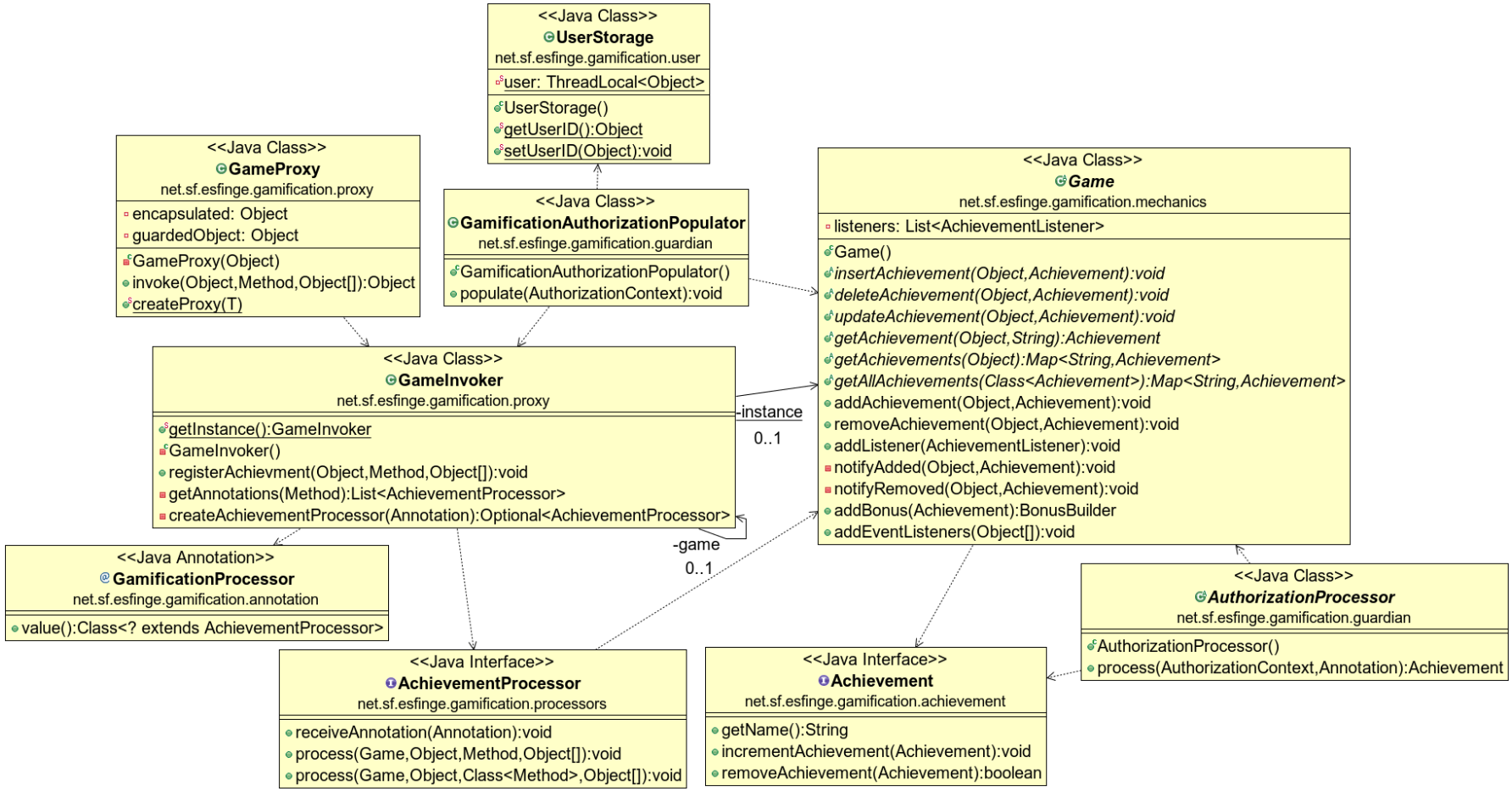
Fonte - Produção do autor

Desta forma a responsabilidade de execução é passada para o Esfinge Guardian e as validações ocorrem de acordo com o diagrama de sequência exibido na Figura 35.

3.4 Diagrama de classes

O diagrama de classes exibido na Figura 45 evidencia as mudanças feitas. Nele é possível verificar que a Classe GameProxy agora possui dois objetos encapsulados, o objeto que foi recebido para criação do PD (encapsulated) e o criado para ser interceptado pelo Esfinge Guardian (guardedObject). Também é possível visualizar que existem 2 novas classes, AuthorizationProcessor e GamificationAuthorizationPopulator. A responsabilidade da classe AuthorizationProcessor é realizar validações na anotação de segurança recuperada pelo Esfinge Guardian e recuperar o Achievement necessário para a autorização. A classe GamificationAuthorizationPopulator tem como responsabilidade a recuperação e inserção das informações necessárias para a autorização, que são a instância especializada de Game escolhida para persistência e o usuário definido para ser utilizado pelo Esfinge Gamification em suas interceptações.

Figura 45 - Diagrama de classes atual Esfinge Gamification

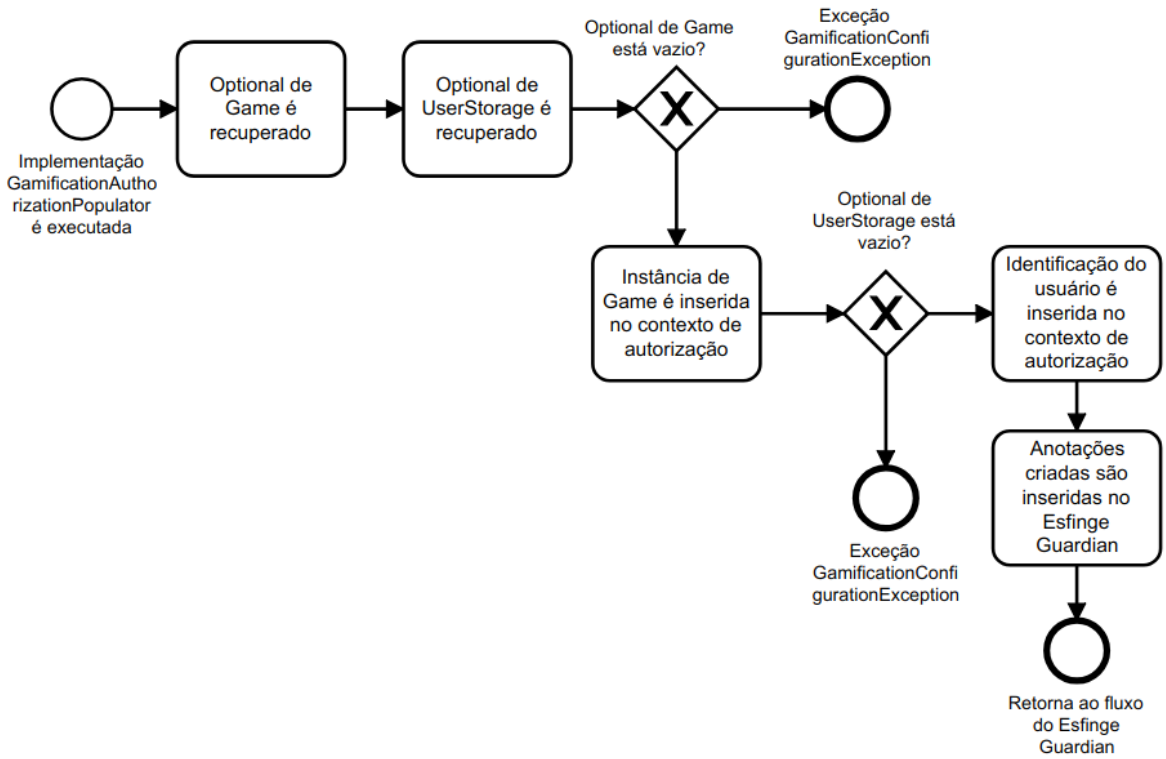


Fonte - Produção do autor

3.5 Integração

Durante o fluxo de execução do Esfinge Guardian (Figura 35), implementações da interface Populator são procuradas para recuperação de informações necessárias para autorização. A implementação criada para recuperação dos dados de gamificação foi chamada de GamificationAuthorizationPopulator conforme citado anteriormente, seu funcionamento é detalhado na Figura 46. Segundo a seção 2.4.2.2 onde o processo de extensão do Esfinge Guardian foi abordado, as implementações de Populator precisariam ser expostas de alguma forma para o *framework*. O método de configuração escolhido foi via arquivo de configuração, portanto existe um arquivo chamado org.esfinge.guardian.populator.Populator com o conteúdo net.sf.esfinge.gamification.guardian.GamificationAuthorizationPopulator localizado no diretório resources/META-INF/services.

Figura 46 - Fluxo de funcionamento GamificationAuthorizationPopulator



Fonte - Produção do autor

A classe Game e o valor armazenado em UserStorage são inseridos no contexto de autorização conforme já dito, pois com o valor armazenado em UserStorage é possível identificar qual usuário terá o Achievement recuperado, e com a classe Game é possível recuperar o Achievement do usuário. As anotações de segurança são inseridas no Esfinge Guardian para que estas sejam identificadas posteriormente, quando as implementações da interface Authorizer forem executadas pelo *framework* conforme Figura 35.

3.6 Funcionalidades

Para utilização das funcionalidades de autorização integradas, as anotações disponíveis no pacote net.sf.esfinge.gamification.annotation.auth exibidas na Tabela 4 foram criadas.

Tabela 4 - Anotações de autorização desenvolvidas

Anotação	Comportamento
@AllowPointGreaterThan, @AllowPointLessOrEqualsThan, @DenyPointLessOrEqualsThan, @DenyPointGreaterThan	Estas anotações verificam se os pontos respeitam as restrições de maior ou menor igual a uma determinada quantidade definida na anotação para que a autorização seja concedida.
@AllowRanking, @AllowLevel, @AllowRankingAndLevel, @AllowRankingOrLevel, @DenyLevel, @DenyRanking, @DenyRankingAndLevel, @DenyRankingOrLevel	As anotações de Ranking verificam as condições das propriedades ranking e level das conquistas, para permitir ou não o acesso a recursos.
@AllowTrophy, @DenyTrophy	Anotações de Trophy permitem ou não que um usuário acesse o recurso.
@AllowReward, @DenyReward	Anotações de Reward verificam se o usuário poderá ou não

	acessar um recurso com determinado Reward.

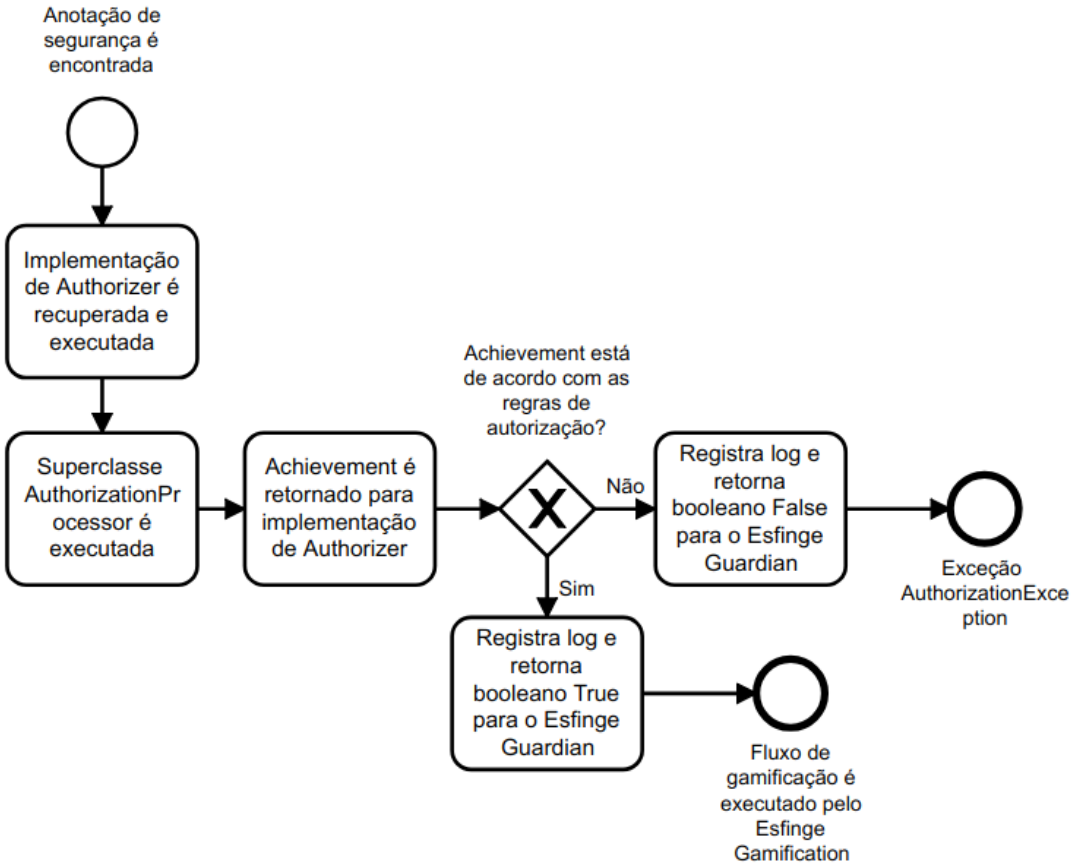
Fonte - Produção do autor

A partir destas anotações é possível adicionar metadados em métodos que necessitam de autorização para execução.

3.7 Processo de autorização

Para cada anotação criada foi preciso implementar a interface Authorizer conforme descrito anteriormente. Na Figura 47 é possível visualizar o processo realizado pelo Esfinge Guardian para verificar as informações de gamificação do usuário.

Figura 47 - Processo de autorização baseado nas conquistas



Fonte - Produção do autor

Todas as implementações de Authorizer criadas estendem a classe abstrata AuthorizationProcessor (Figura 48), que tem como objetivo principal a recuperação do Achievement utilizado na autorização e a validação da anotação de segurança recebida conforme já dito. É possível verificar nas linhas 5 e 6 que o Game e o usuário inseridos no contexto de autorização pela classe GamificationAuthorizationPopulator (Figura 46) são utilizados nesta etapa, pois o Achievement é buscado e recuperado com estas informações na linha 25.

Figura 48 - Classe AuthorizationProcessor

```
1  public abstract class AuthorizationProcessor {
2
3      public Achievement process(AuthorizationContext context, Annotation
           securityAnnotation) {
4
5          Game game = (Game) context.getEnvironment().get("game");
6          Object user = (Object) context.getResource().get("currentUser");
7
8          if (Objects.isNull(securityAnnotation))
9              throw new GamificationConfigurationException(
10                     "One security annotation it's necessary to validade
                           this process");
11
12         Class<? extends Annotation> annotationType =
               securityAnnotation.annotationType();
13         String achiev = "";
14
15         try {
16             Method achievGetter =
                   annotationType.getMethod("achievementName");
17             achiev = (String) achievGetter.invoke(securityAnnotation);
18         } catch (IllegalAccessException | IllegalArgumentException |
               InvocationTargetException | NoSuchMethodException
19                 | SecurityException invokeException) {
20             throw new GamificationConfigurationException(
21                     "Achievement name property could not be found in
                           annotation " + annotationType.getName(),
22                     invokeException);
23         }
24
25         Achievement achievement = game.getAchievement(user, achiev);
26
27         return achievement;
28     }
29 }
```

Fonte - Produção do autor

Na linha 8 a anotação de autorização recebida é validada para que exceção NullPointerException não ocorra em tempo de execução quando o tipo da anotação for recuperado via reflexão (linha 12). Como as implementações de Achievement presentes no Esfinge Gamification devem possuir um método para recuperar seu nome (Figura 19) foi criada a convenção de código "achievementName" para as anotações de segurança, isto é, toda anotação de segurança possui uma propriedade chamada "achievementName()", possibilitando assim a recuperação do nome do Achievement via reflexão, processo realizado nas linhas 16 e 17.

Com essas implementações foi possível realizar a integração dos *frameworks* e o controle de acesso baseado em conquistas cumprindo os objetivos definidos.

4 ANÁLISES E RESULTADOS

Neste capítulo serão apresentados os testes implementados com a solução desenvolvida, a fim de avaliar o funcionamento a partir de duas perguntas de pesquisa (PP):

- PP1: O acoplamento entre o *framework* e a aplicação é baixo?
- PP2: O *framework* é capaz de realizar o controle de acesso baseado em conquistas?

4.1 Metodologia

Para o desenvolvimento e testes do estudo de caso é necessário que a aplicação criada possua controle de acesso em funcionalidades, incremento de pontos e adição de Achievements em determinadas situações. Quando algum erro ocorrer mensagens descritivas deverão ser exibidas. Os testes serão avaliados a partir de *logs* da aplicação, onde mensagens de sucesso e falha são exibidos.

4.2 Aplicação desenvolvida

Foi desenvolvida uma aplicação *web*[5] para utilização dos recursos desenvolvidos, com o objetivo de compartilhamento de ideias e conversas sobre assuntos gerais. Esta aplicação utiliza os mecanismos de pontos e troféus para incentivar a interação entre os usuários, a partir destes mecanismos as regras da Figura 49 foram definidas. Quando usuários conseguem ultrapassar as restrições impostas pelas regras novas funcionalidades são desbloqueadas conforme a utilização da aplicação, e assim, consequentemente, os usuários interagem entre si.

[5] O estudo de caso está disponível no endereço https://forum-tg.netlify.com

Figura 49 - Regras definidas para a aplicação web

Fonte – Produção do autor

Esta aplicação possui o *back-end* hospedado no Heroku[6], os próximos testes abordarão os registros da aplicação hospedada no Heroku para confirmar que o Esfinge Guardian realizou as validações esperadas e pode ser utilizado em produção.

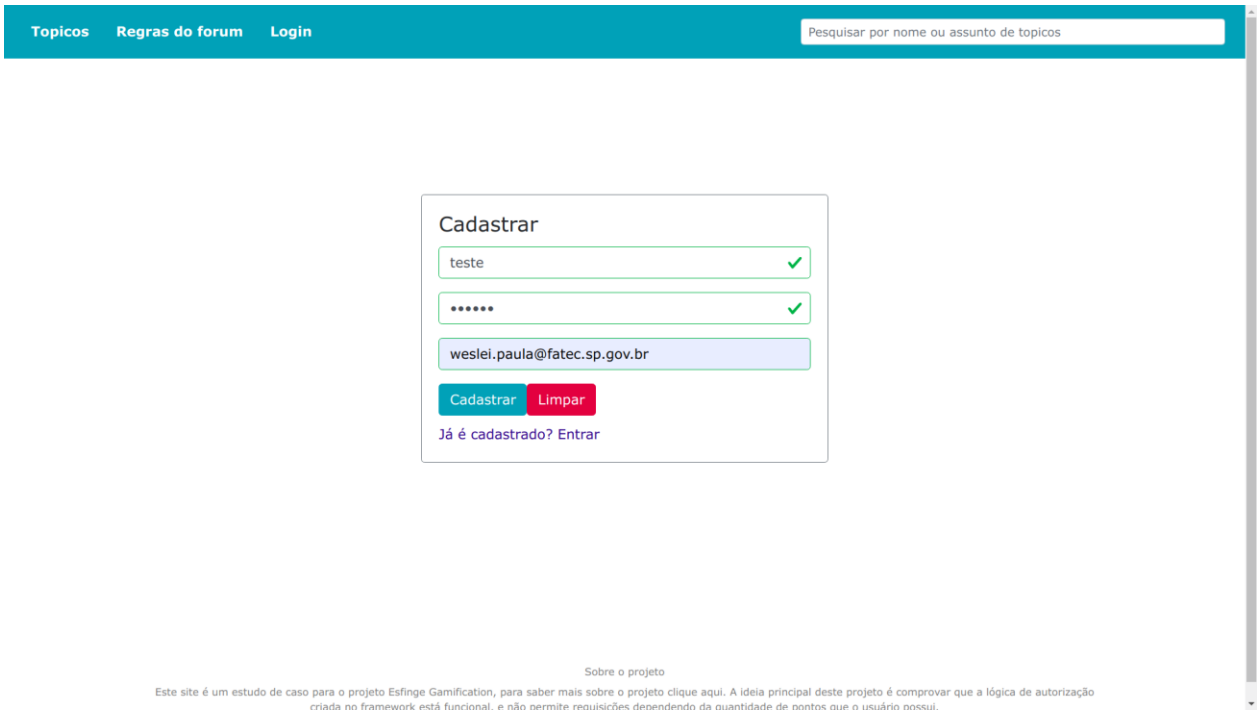
4.3 Fluxo de funcionamento

O primeiro caso de teste tem como objetivo mostrar que o Esfinge Guardian não interfere no fluxo de funcionamento do Esfinge Gamification quando não há anotações de segurança, permitindo que os recursos de gamificação sejam utilizados normalmente. Para validar essa premissa, um tópico será adicionado com o usuário teste. Quando o usuário acessar seu perfil poderá visualizar que possui agora 10 pontos.

Na primeira etapa o usuário teste será criado no *site* para que não possua nenhum ponto, a Figura 50 exibe o formulário de cadastro com as informações preenchidas.

[6] Heroku é um PaaS que significa Platform as a Service**.** A aplicação está disponível no endereço https://forum-back-end.herokuapp.com.

Figura 50 - Cadastro no fórum



Fonte - Produção do autor

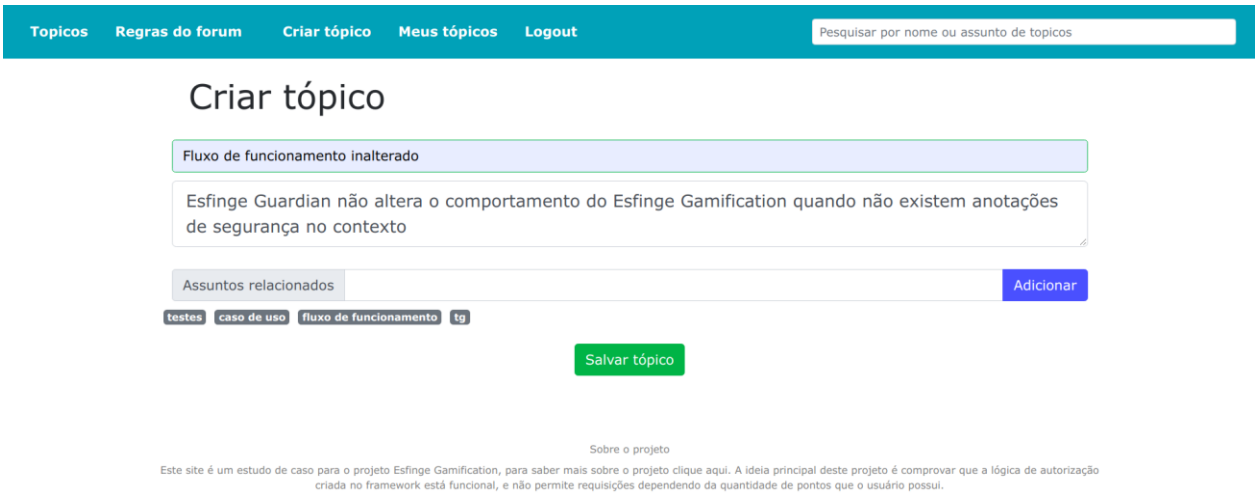
Após o cadastro realizado, o usuário é redirecionado para a página inicial (Figura 51) e clica em "Criar tópico".

Figura 51 - Tela incial após cadastro

Fonte - Produção do autor

Nesta tela, outro formulário é exibido para ser preenchido com as informações relacionadas ao assunto desejado. A Figura 52 exibe como este formulário foi preenchido.

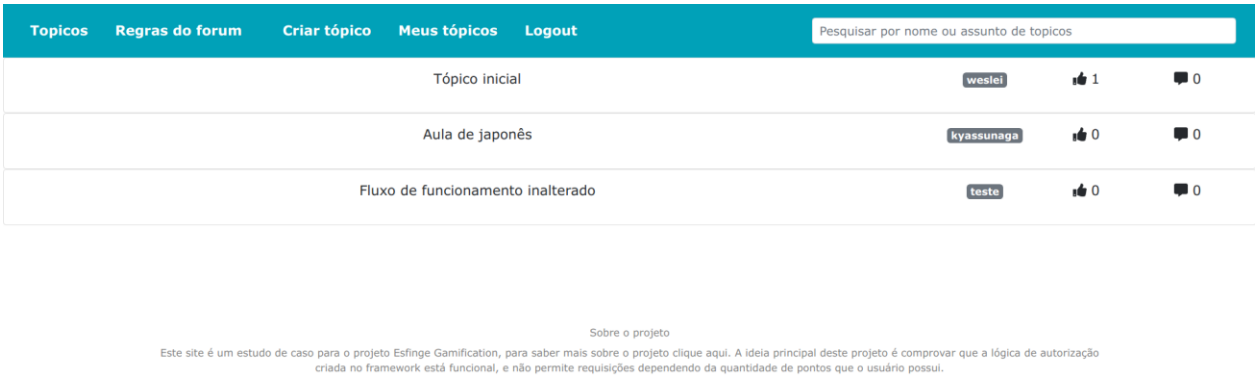
Figura 52 - Primeiro tópico criado pelo usuário teste



Fonte - Produção do autor

Após a criação do tópico o usuário é redirecionado para a tela inicial novamente, e seu tópico já está disponível conforme a Figura 53 evidencia.

Figura 53 - Tópico criado pelo usuário teste



Fonte - Produção do autor

A tela do perfil de usuários é exibida quando os botões cinzas com o nome do usuário que criou o tópico são clicados, estes botões ficam disponíveis também em comentários. A Figura 54 confirma a premissa deste teste, exibindo o perfil do usuário teste com 10 pontos após a criação do tópico.

Figura 54 - Resultado obtido após a criação do tópico

Fonte - Produção do autor

Com as evidências exibidas acima ainda não é possível fazer a identificação de que validações de segurança não foram executadas. Para realizar isto foi preciso verificar os *logs* do servidor. A Figura 55 exibe estes *logs* separados por cores verdes, isto é, a cada linha com o início verde onde a data é exibida, um novo *log* foi registrado. O primeiro *log* são registros do clique no botão "Salvar" (Figura 52), onde uma requisição do tipo POST é feita para que o tópico seja salvo. O segundo *log* é o redirecionamento para a página inicial (Figura 53), onde uma requisição GET é feita para recuperação dos tópicos existentes.

Figura 55 - *Logs* do Heroku não exibindo traços de execução do Esfinge Guardian

```
2019-05-02T19:31:05.890068+00:00 heroku[router]: at=info method=POST path="/topics" host=forum-back-
end.herokuapp.com request_id=e9f76471-7de3-4c3f-a02a-ece735c5950c fwd="191.23.52.112" dyno=web.1 con
nect=0ms service=115ms status=201 bytes=1149 protocol=https
2019-05-02T19:31:05.787733+00:00 app[web.1]: @net.sf.esfinge.metadata.annotation.validator.NotNull()
2019-05-02T19:31:05.787955+00:00 app[web.1]: @net.sf.esfinge.metadata.annotation.validator.NotNull()
2019-05-02T19:31:05.788127+00:00 app[web.1]: @net.sf.esfinge.metadata.annotation.validator.NotNull()
2019-05-02T19:31:05.788301+00:00 app[web.1]: @net.sf.esfinge.metadata.annotation.validator.NotNull()
2019-05-02T19:31:05.789427+00:00 app[web.1]: @net.sf.esfinge.metadata.annotation.validator.NotNull()
2019-05-02T19:31:05.800301+00:00 app[web.1]: 2019-05-02 19:31:05.800  INFO 4 --- [io-30413-exec-5] o
rg.reflections.Reflections             : Reflections took 9 ms to scan 1 urls, producing 4 keys and
 64 values
2019-05-02T19:31:05.845981+00:00 app[web.1]: 2019-05-02 19:31:05.845  INFO 4 --- [io-30413-exec-5] o
rg.reflections.Reflections             : Reflections took 3 ms to scan 1 urls, producing 1 keys and
 4 values
2019-05-02T19:31:07.656814+00:00 heroku[router]: at=info method=GET path="/topics" host=forum-back-e
nd.herokuapp.com request_id=38a1796b-45e0-4227-86da-59b62893e91f fwd="191.23.52.112" dyno=web.1 conn
ect=1ms service=29ms status=200 bytes=1712 protocol=https
```

Fonte - Produção do autor

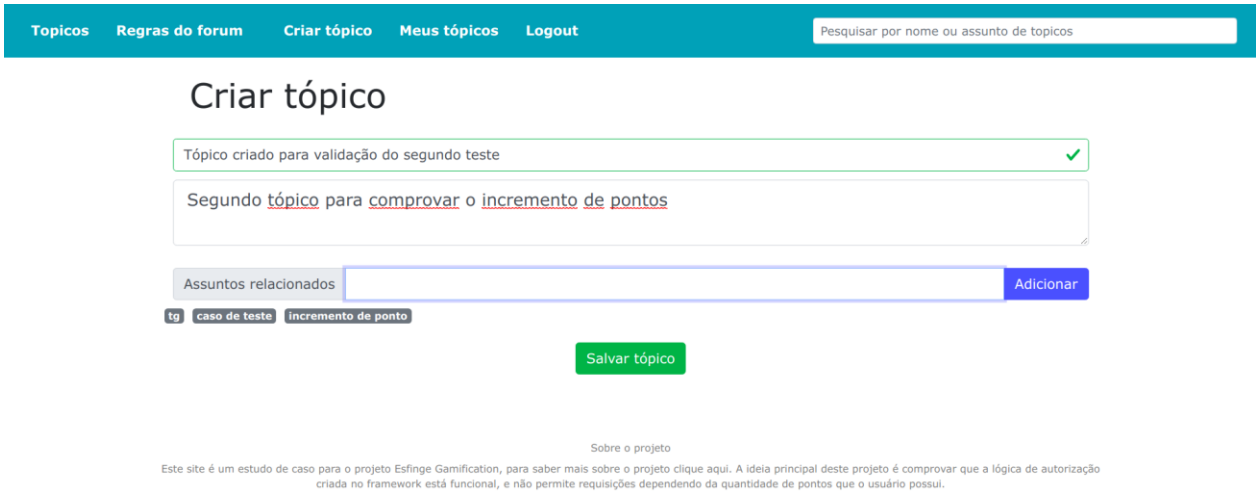
De acordo com o previsto na premissa deste teste o usuário recebeu os 10 pontos após a criação do tópico e o Esfinge Guardian não executou nenhuma validação de segurança.

4.4 Incremento dos pontos

Este teste tem como objetivo a validação do incremento dos pontos após a criação de tópicos no *site*, para que as restrições possam ser atingidas e novos recursos desbloqueados. Isto será comprovado a partir do usuário teste utilizado anteriormente, este possui 10 pontos e criará outro tópico, passando a possuir 20 pontos.

O mesmo processo realizado na Figura 52 foi realizado, porém o tópico exibido na Figura 56 foi criado.

Figura 56 - Formulário preenchido com novo tópico



Fonte - Produção do autor

Retornando ao perfil do usuário é possível verificar que sua pontuação agora é 20, conforme Figura 57.

Figura 57 - Pontuação do usuário teste

Fonte - Produção do autor

O objetivo deste teste também foi atingido, comprovando o incremento das conquistas permitindo assim que as restrições sejam eliminadas conforme a utilização.

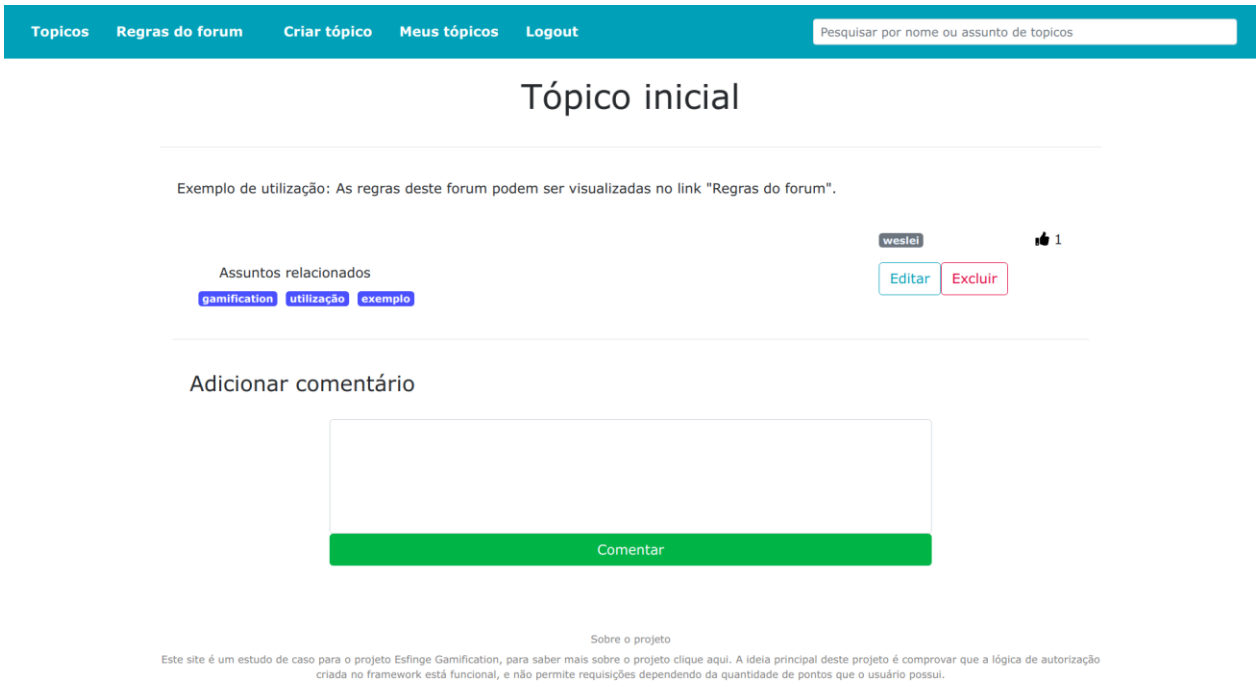
4.5 Usuário sem permissão

As restrições serão validadas neste caso de teste, em específico a anotação @AllowPointGreaterThan, que tem como objetivo controlar o acesso de usuários baseando-se em seus pontos, conforme citado na Tabela 4.

Quando um usuário possuir menos de 50 pontos, de acordo com as regras do fórum (Figura 49) este não deve conseguir concluir o processo de atualização do tópico. Para realizar este teste o usuário weslei foi definido, ele possui 10 pontos atualmente, tendo criado apenas um tópico no *site*. O conteúdo do tópico será alterado e uma requisição do tipo PUT será realizada ao servidor para salvar as alterações. O usuário deverá receber a mensagem de erro "Você não possui permissão para atualizar tópicos!" devido à falta de pontos.

Na Figura 58 o usuário está na página de visualização detalhada do tópico. É importante ressaltar que os botões "Editar" e "Excluir" ficam disponíveis apenas para o proprietário do conteúdo.

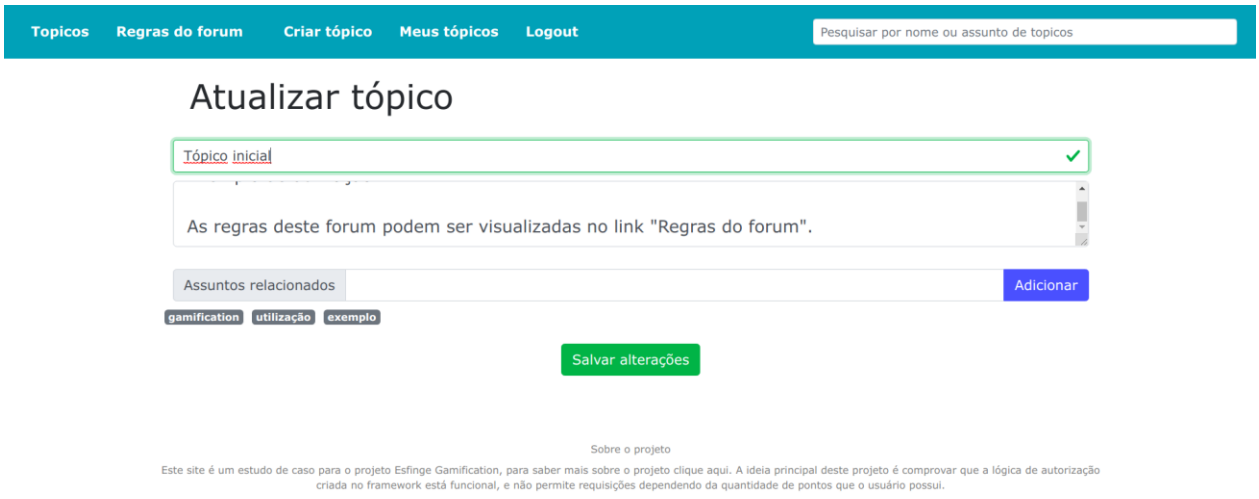
Figura 58 -Tela de visualização detalhada de tópicos



Fonte - Produção do autor

Quando o botão "Editar" é selecionado os campos do tópico tornam-se editáveis vide Figura 59.

Figura 59 - Conteúdo do tópico editável



Fonte - Produção do autor

O botão "Salvar alterações" é selecionado pelo usuário e consequentemente uma requisição PUT é feita ao servidor para salvar o conteúdo modificado. As validações são feitas pelo *back-end* e a mensagem de erro "Você não possui permissão para atualizar tópicos!" é exibida ao usuário conforme Figura 60.

Figura 60 - Mensagem de erro exibida ao realizar processo de edição de tópico sem autorização

Fonte - Produção do autor

Seguindo o raciocínio do primeiro teste, apenas a mensagem de erro no *front-end* não é suficiente para validação. Na Figura 47 é citada a criação de *logs* durante a validação do Esfinge Guardian. Estes *logs* são criados justamente para identificação de quando acessos a recursos protegidos são feitos. A Figura 61 exibe o *log* registrado no momento da requisição inválida realizada pelo usuário weslei, validando o teste.

Figura 61 - *Log* de falta de autorização ao realizar operação de atualização de tópico

```
2019-05-02T22:38:57.716176+00:00 heroku[router]: at=info method=PUT path="/topics/5cafcf259323840004
5c5699" host=forum-back-end.herokuapp.com request_id=9b141c04-4cdc-4bba-9200-ed9f37c92e77 fwd="191.2
3.52.112" dyno=web.1 connect=1ms service=580ms status=403 bytes=722 protocol=https
2019-05-02T22:38:57.608472+00:00 app[web.1]: 2019-05-02 22:38:57.608  INFO 4 --- [o-48471-exec-10] o
rg.reflections.Reflections                : Reflections took 14 ms to scan 1 urls, producing 4 keys an
d 64 values
2019-05-02T22:38:57.640759+00:00 app[web.1]: 2019-05-02 22:38:57.640  INFO 4 --- [o-48471-exec-10] o
rg.reflections.Reflections                : Reflections took 4 ms to scan 1 urls, producing 1 keys and
 4 values
2019-05-02T22:38:57.692334+00:00 app[web.1]: 2019-05-02 22:38:57.692  INFO 4 --- [o-48471-exec-10] o
rg.mongodb.driver.connection              : Opened connection [connectionId{localValue:8, serverValue:
212813}] to forum-shard-00-01-ykkey.mongodb.net:27017
2019-05-02T22:38:57.702573+00:00 app[web.1]: 2019-05-02 22:38:57.702  WARN 4 --- [o-48471-exec-10] .
g.g.a.p.AllowPointGreaterThanAuthorizer : Unauthorized accesss: Required achievement: Point points R
equired quantity: 50 User's quantity: 10
```

Fonte - Produção do autor

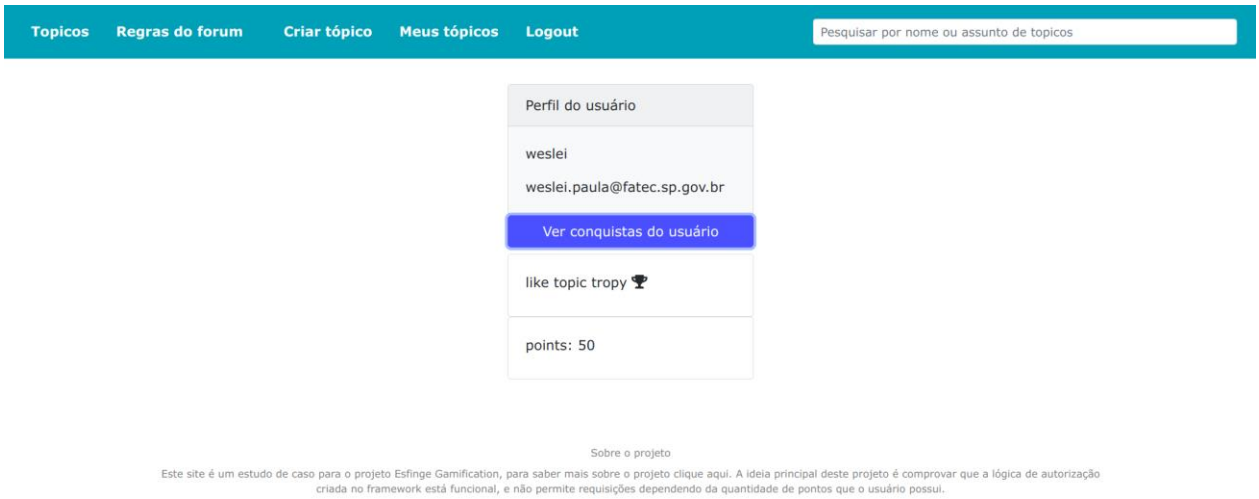
Com a criação de *logs* no Esfinge Gamification foi possível confirmar as premissas definidas neste teste, onde o usuário weslei tentou alterar um tópico criado, porém não possuía pontos suficientes para realizar esta operação.

4.6 Usuário com permissão

O objetivo deste caso de teste é a autorização, neste caso o processo feito indevidamente no teste anterior será executado novamente com o usuário weslei, porém este possuirá 50 pontos e tentará alterar o mesmo tópico. É esperado que as alterações sejam realizadas com sucesso e que *logs* sejam registrados pelo servidor para confirmação da validação do Esfinge Guardian.

Como o incremento das conquistas já foi testado e para agilizar o processo da criação do teste, o ambiente foi alterado para atender este requisito, conforma Figura 60 exibe.

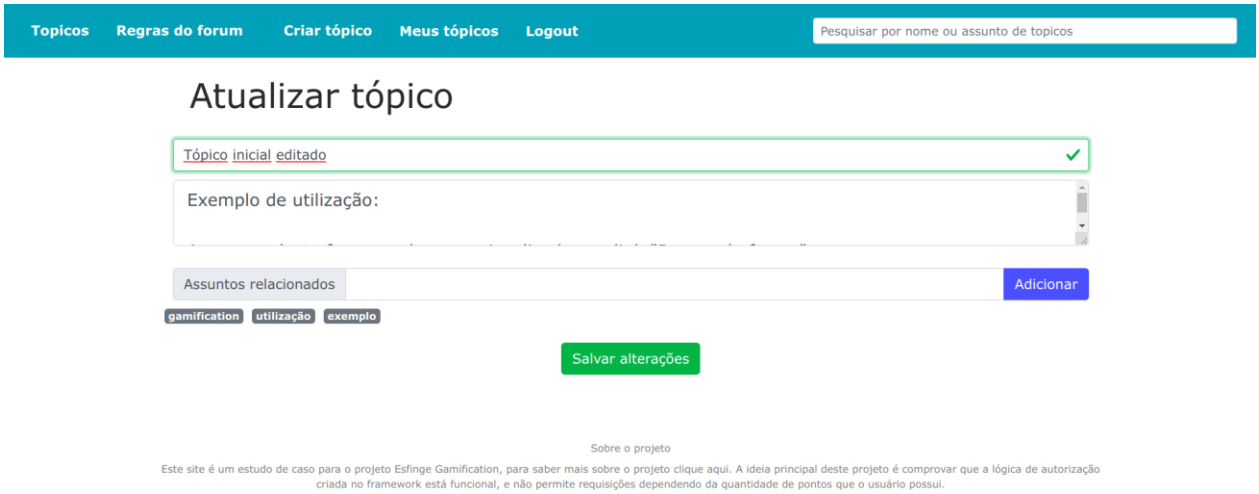
Figura 62 - Pontuação do usuário weslei



Fonte - Produção do autor

Novamente o tópico foi selecionado e alterado conforme Figura 63.

Figura 63 - Alterações realizadas no tópico



Fonte - Produção do autor

O botão "Salvar alterações" é selecionado e o tópico é alterado com sucesso conforme exibido na Figura 63.

Figura 64 - Pontuação do usuário weslei

Produção do autor

No servidor é possível identificar a verificação realizada pelo Esfinge Guardian. A Figura 65 exibe os *logs* registrados.

Figura 65 - Pontuação do usuário weslei

```
2019-05-02T23:35:27.607980+00:00 app[web.1]: @net.sf.esfinge.metadata.annotation.validator.NotNull()
2019-05-02T23:35:27.608177+00:00 app[web.1]: @net.sf.esfinge.metadata.annotation.validator.NotNull()
2019-05-02T23:35:27.608366+00:00 app[web.1]: @net.sf.esfinge.metadata.annotation.validator.NotNull()
2019-05-02T23:35:27.608510+00:00 app[web.1]: @net.sf.esfinge.metadata.annotation.validator.NotNull()
2019-05-02T23:35:27.617030+00:00 app[web.1]: @net.sf.esfinge.metadata.annotation.validator.NotNull()
2019-05-02T23:35:27.855244+00:00 app[web.1]: 2019-05-02 23:35:27.854  INFO 4 --- [io-19134-exec-3] o
rg.reflections.Reflections                : Reflections took 21 ms to scan 1 urls, producing 4 keys an
d 64 values
2019-05-02T23:35:27.899718+00:00 app[web.1]: 2019-05-02 23:35:27.899  INFO 4 --- [io-19134-exec-3] o
rg.reflections.Reflections                : Reflections took 8 ms to scan 1 urls, producing 1 keys and
 4 values
2019-05-02T23:35:27.976474+00:00 app[web.1]: 2019-05-02 23:35:27.976  INFO 4 --- [io-19134-exec-3] o
rg.mongodb.driver.connection              : Opened connection [connectionId{localValue:8, serverValue:
209414}] to forum-shard-00-01-ykkey.mongodb.net:27017
2019-05-02T23:35:27.988472+00:00 app[web.1]: 2019-05-02 23:35:27.988  INFO 4 --- [io-19134-exec-3] .
g.g.a.p.AllowPointGreaterThanAuthorizer : Authorized accesss: Required achievement: Point points Qua
ntity: 50 User's quantity: 50
2019-05-02T23:35:28.061537+00:00 heroku[router]: at=info method=PUT path="/topics/5cafcf259323840004
5c5699" host=forum-back-end.herokuapp.com request_id=4efadda2-e7b1-44cb-b542-f202c9a46101 fwd="191.2
3.52.112" dyno=web.1 connect=0ms service=958ms status=200 bytes=1112 protocol=https
```

Fonte - Produção do autor

Como esperado, o servidor registrou *logs* validando a requisição feita pelo usuário weslei, comprovando que o Esfinge Guardian analisou os dados de gamificação para autorizar a requisição. Um ponto interessante deste *log*, é que a mensagem retornada pelo Esfinge Guardian referente a requisição PUT é exibida antes do *log* da própria requisição, o que pode gerar confusão em um primeiro momento de análise, entretanto o *log* de validação registrado pelo Esfinge Guardian é relativo ao processo de autorização desencadeado pela requisição PUT, realizada pelo usuário weslei.

4.7 Matriz de Dependência Estrutural

Matriz de Dependência Estrutural, do inglês *Dependency Structure Matrix* (DSM), tem como objetivo facilitar a visualização entre as dependências do projeto, o acoplamento entre as classes (BROWNING, 2001). Neste caso o que será abordado é o acoplamento envolvendo a aplicação e o Esfinge Gamification, a partir das Figuras 66 e 67.

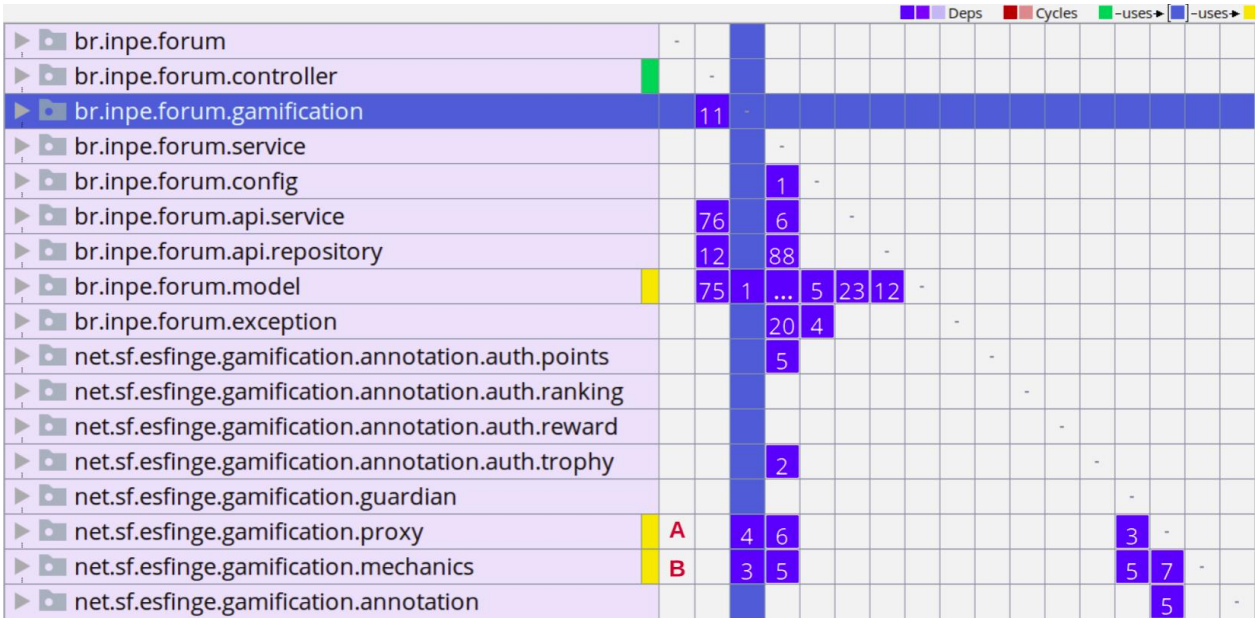
A Figura 66 exibe o acoplamento envolvendo a configuração do *framework*, esta é realizada na classe GameSetup, presente no pacote br.inpe.forum.gamification. É possível visualizar que esta classe depende de 2 módulos do *framework*: 1) Para criação e interceptação do PD (A); 2) Para definição de persistência de dados (B). Estas dependências são identificadas em amarelo após o nome dos pacotes. Já as classes que dependem de GameSetup são os *controllers* da aplicação, devido a sua responsabilidade, que é a tratativa das requisições.

Este foi o ponto ideal para interceptações, pois quando *controllers* invocam a lógica de negócios na camada de serviço (pacote br.inpe.forum.service), a invocação é interceptada e os metadados definidos nas classes de serviço são lidos pelo *framework*, que realiza o comportamento definido no metadado, conforme exibido nos testes anteriores.

A Figura 67 apresenta a DSM do pacote de serviço, onde as definições de metadados são adicionadas. Como metadados de controle de acesso foram definidos nestas classes, as anotações de segurança (A, B) viram dependências. As dependências C e D existem, pois foi implementado um serviço que recupera dados de gamificação de usuários, portanto é preciso recuperar a

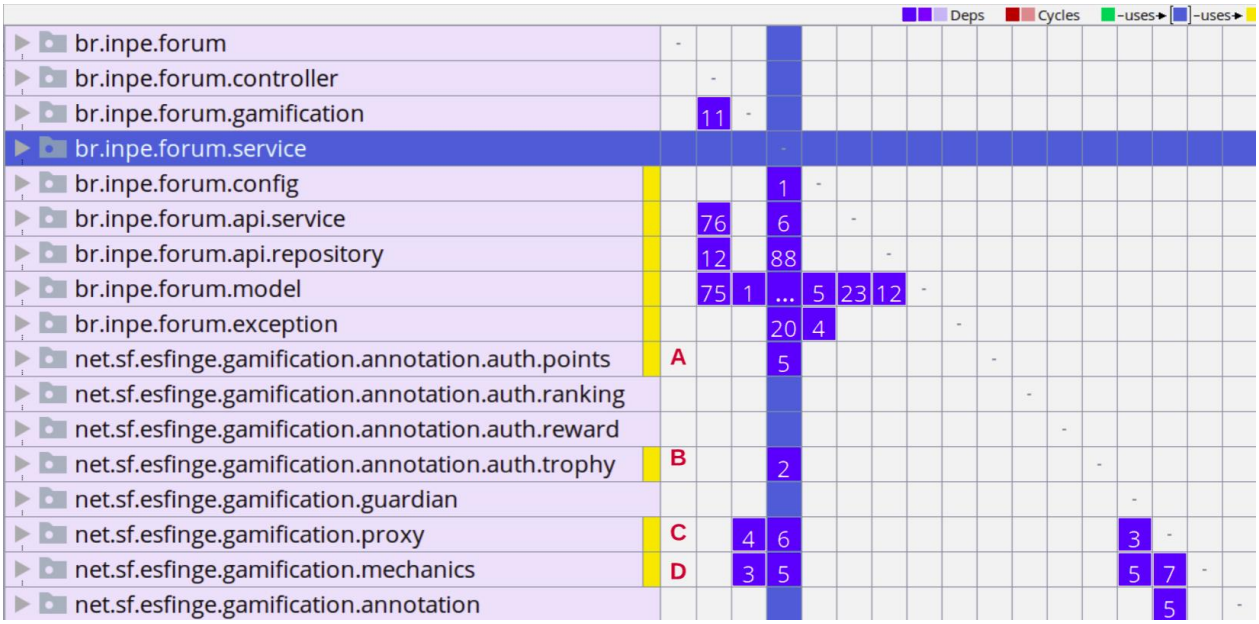
instância de Game que é gerenciada pela classe GameProxy (C), e assim recuperar os dados armazenados com a especialização de Game (D).

Figura 66 -DSM de configuração do *framework*



Fonte - Produção do autor

Figura 67 - DSM classes serviço



Fonte - Produção do autor

Baseado nas DSMs apresentadas é possível responder a PP1 positivamente, pois o *framework* possui baixo acoplamento com a solução desenvolvida, sendo utilizado apenas em uma classe para configuração, esta é utilizada por classes com responsabilidade de *controller*, para que os métodos possam ser interceptados e pelo serviço responsável por recuperar informações de gamificação de usuários. A principal interface entre o *framework* e a aplicação desenvolvida são as anotações que adicionam os metadados necessários.

A respeito da PP2 é possível concluir que a integração foi um sucesso e que o controle de acesso é eficaz devido aos testes executados nesta seção. Todas as execuções que precisariam ser validadas pelo Esfinge Guardian foram registradas com sucesso pelo servidor a partir de *logs* que confirmaram esta validação.

Desta forma é possível dizer que o Esfinge Gamification é desacoplado de domínio, pode ser facilmente inserido como uma dependência e utilizado em diversas aplicações sem dificuldades, devido a sua generalização.

5 CONCLUSÕES

A solução proposta para a falta de recursos de controle de acesso no Esfinge Gamification foi sua integração com o *framework* Esfinge Guardian, que possui tais funcionalidades; para isto foi necessário utilizar os pontos de extensão presentes no *framework* de segurança. Desta forma foi possível realizar a criação das funcionalidades de controle de acesso baseado em conquistas que podem ser utilizadas via metadados em aplicações Java.

5.1 Contribuições do Trabalho

- Anotações de controle de acesso baseado em conquistas;
- Integração entre o Esfinge Gamification e Esfinge Guardian;
- API extensiva para criação de novas regras de autorização;
- Implementação de persistência para o MongoDB;
- Método para recuperar todas as conquistas por tipo;
- Dependências do Esfinge Guardian foram configuradas no pom.xml do projeto para *download* automático das dependências transitivas pelo Maven;
- Esfinge Guardian foi configurado para possuir um Super POM e módulos;
- Esfinge Guardian e Esfinge Gamification foram publicados no Maven Central para *download*. Esfinge Guardian com as novas configurações e Esfinge Gamification foi atualizado com as funcionalidades de autorização;
- Artigo Ferramenta de controle de acesso baseado em gamificação (PINTO et al., 2019) foi publicado na CBSOFT 2019 na sessão de ferramentas.
- Documentação do projeto foi atualizada.

5.2 Dificuldades

Entender que códigos são dados e podem ser interpretados e processados não é algo trivial, assim a compreensão do paradigma reflexivo gera confusão em um primeiro momento, pela inversão de pensamento que precisa existir (Figura 5), muitas vezes parâmetros recebidos

em métodos são classes, e a partir disso a ação desejada ocorre. Ter noção de quando usar reflexão é algo interessante, pois como é um recurso caro computacionalmente falando, quando se deseja alta performance, esta não é uma boa alternativa.

A estrutura de diretórios referente as implementações de Populator foi uma dificuldade devido ao não armazenamento destes arquivos na exportação do projeto Esfinge Guardian. Desta forma era necessário criar a árvore de diretórios em todos os projetos novamente, porém quando estes eram exportados não funcionavam conforme o esperado. Este problema foi solucionado com configurações criadas no arquivo de configuração pom.xml, onde ficam armazenadas as dependências e configurações de projetos com o gerenciador de dependências Maven. Após esta configuração foi verificado que apenas alterando a estrutura de diretórios de acordo com a Figura 31 os arquivos seriam exportados com o projeto.

Para recuperar propriedades em arquivos o Esfinge Guardian utilizava uma implementação de ResourceBundle. Este foi um problema devido a seu propósito, pois a classe ResourceBundle tem como objetivo prover recursos variáveis para localizações, ou seja, de acordo com a localização do usuário diferentes traduções de uma palavra podem ser retornadas por exemplo (ORACLE, a). A implementação foi alterada para a classe Properties que atende o objetivo desejado, que é a recuperação de propriedades em arquivos (ORACLE, b).

5.3 Trabalhos futuros

- Seria interessante o método que recupera todas as conquistas por tipo existente no *framework* possuir uma forma de ordenação, para no caso da criação de rankings de posições por exemplo;
- Adição de funcionalidades de bônus, isto é, quando os bônus são executados o dobro de pontos ou conquistas diferentes são recebidas;
- Adição de anotações de eventos, pois atualmente o Esfinge Gamification possui duas anotações para eventos.
- Padronização da localização das anotações, pois o Esfinge Gamification recupera metadados existentes em intefaces e o Esfinge Guardian em classes.

- Integração contínua e entrega contínua no Maven Central, para que quando novas versões serem lançadas fiquem disponível para desenvolvedores.

REFERÊNCIAS BIBLIOGRÁFICAS

ARNOLD, Ken; GOSLING, James; HOLMES, David. *The Java programming language*. [S.l.]: Addison Wesley Professional, 2005.

BARTSCH, Steffen. Authorization enforcement usability case study. In: SPRINGER. *International Symposium on Engineering Secure Software and Systems*. [S.l.], 2011. p.209–220.

BLOCH, Joshua et al. Jsr 175: A metadata facility for the java programming language. 2004.

BROWNING, Tyson R. Applying the design structure matrix to system decompositionand integration problems: a review and new directions. *IEEE Transactions on Engineeringmanagement*, IEEE, v. 48, n. 3, p. 292–306, 2001.

BUCKLEY, Alex.JSR 175: *A Metadata Facility for the JavaTM Programming Language*. 2002. Disponível em: <https://www.jcp.org/en/jsr/detail?id=175>.

BURKE, Katie. Behind the scenes of foldit, pioneering science gamification. *American Scientist*, v. 100, n. 6, p. 1, 2012.

DARCY, Joe. *JCP 269*. 2005. Disponível em: <https://www.jcp.org/en/jsr/detail?id=269>.

DETERDING, Sebastian; SICART, Miguel; NACKE, Lennart; O'HARA, Kenton; DIXON, Dan. Gamification. using game-design elements in non-gaming contexts. In:ACM. *CHI'11 extended abstracts on human factors in computing systems*. [S.l.], 2011. p.2425–2428.

FOUNDATION, Apache Software. *Introduction to the POM - Super POM*. 2019. Disponível em: <https://maven.apache.org/guides/introduction/introduction-to-the-pom.html#Super_POM>.

GAMMA, Erich. *Design patterns: elements of reusable object-oriented software.* [S.l.]:Pearson Education India, 1995.

GAMMA, Erich. *Padrões de Projetos: Soluções Reutilizáveis.* [S.l.]: Bookman editora, 2009.

GIERKE, Oliver; DARIMONT, Thomas; STROBL, Christoph; PALUCH, Mark;BRYANT, Jay. Spring framework, 2019. Disponível em: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/>.

GOOGLE. *Search for gamification term*. Disponível em: <https://trends.google.com/trends>.

GROH, Fabian. Gamification: State of the art definition and utilization. *Institute of MediaInformatics Ulm University*, v. 39, p. 31, 2012.

GUERRA, Eduardo. *Componentes reutilizáveis em Java com Reflexão e Anotações.* [S.l.]:Editora Casa do Código, 2014.

GUERRA, Eduardo; FERNANDES, Clovis; SILVEIRA, Fábio Fagundes. Architectural patterns for metadata-based frameworks usage. In: ACM. *Proceedings of the 17th Conference on Pattern Languages of Programs*. [S.l.], 2010. p. 4.

GUERRA, Eduardo Martins. A conceptual model for metadata-based frameworks. *Aeronautics Institute of Technology, São José dos Campos*, 2010.

GUERRA, Eduardo Martins; CANDIA, Marcos Paulo Lobo de; SILVA, Jeremias Poncianoda; SOUZA, Hugo Alexandro; SANTOS, Lucas Shen dos; EMYGDIO, Danielle; PORTO, Sandy; JúNIOR, José Lázaro de Siqueira. *Esfinge Gamification*. 2011. Disponível em: <http://esfinge.sourceforge.net/Gamification.html>.

GUERRA, Eduardo Martins; CANDIA, Marcos Paulo Lobo de; SILVA, Jeremias Poncianoda; SOUZA, Hugo Alexandro; SANTOS, Lucas Shen dos; EMYGDIO, Danielle; PORTO, Sandy; JÚNIOR, José Lázaro de Siqueira. *Esfinge Project*. 2011. Disponível em: <http://esfinge.sf.net>.

GUERRA, Eduardo M; FORNARI, Gabriel; COSTA, Wanderson S; PORTO, Sandy M; CANDIA, Marcos PL; SILVA, Tiago Silva da. An approach for modularizing Gamification concerns. In: SPRINGER. *International Conference on Computational Science and Its Applications*. [S.l.], 2017. p. 635–651.

GUERRA, Eduardo M; SOUZA, Jerffeson T De; FERNANDES, Clovis T. A pattern language for metadata-based frameworks. In: ACM. *Proceedings of the 16th Conference on Pattern Languages of Programs.* [S.l.], 2009. p. 3.

HAMARI, Juho; KOIVISTO, Jonna; SARSA, Harri. Does gamification work?-a literature review of empirical studies on gamification. In: IEEE. *2014 47th Hawaii international conference on system sciences (HICSS)*. [S.l.], 2014. p. 3025–3034.

HU, Vincent C; KUHN, D Richard; FERRAIOLO, David F. Attribute-based accesscontrol. *Computer*, IEEE, v. 48, n. 2, p. 85–88, 2015.

JBOSS. *Hibernate community Documentation - Chapter 2. Mapping Entities*. Disponível em: <https://docs.jboss.org/hibernate/annotations/3.5/reference/en/html/entity.html>.

JBOSS. *Hibernate community Documentation - Overriding metadata through XML*. Disponível em: <https://docs.jboss.org/hibernate/orm/3.3/reference/pt-BR/html/session-configuration.html#configuration-xmlconfig>.

JOHNSON, Ralph E; FOOTE, Brian. Designing reusable classes. *Journal of object-oriented programming*, v. 1, n. 2, p. 22–35, 1988.

JOY, Bill; STEELE, Guy; GOSLING, James; BRACHA, Gilad. *The Java language specification*. [S.l.]: Addison-Wesley Reading, 2000.

LINDQVIST, Hakan. Mandatory access control. *Master's Thesis in Computing Science, Umea University, Department of Computing Science, SE-901*, v. 87, 2006.

LINEHAN, Conor; KIRMAN, Ben; ROCHE, Bryan. Gamification as behavioral psychology. In: *The gameful world: Approaches, issues, applications*. [S.l.]: MIT Press, 2015.

MARKIEWICZ, Marcus Eduardo; LUCENA, Carlos JP de. Object oriented framework development. *Crossroads*, v. 7, n. 4, p. 3–9, 2001.

ORACLE. *Class ResourceBundle*. Disponível em: <https://docs.oracle.com/javase/8/docs/api/java/util/ResourceBundle.html>.

ORACLE. *Class Properties*. Disponível em: <https://docs.oracle.com/javase/8/docs/api/java/util/Properties.html>.

ORACLE. *Java Programming Environment and the Java Runtime Environment (JRE)*. 2010. Disponível em: <https://docs.oracle.com/cd/E19455-01/806-3461/6jck06gqb/index.html>.

OVERVIEW of Spring MVC Architecture. 2013. Disponível em: <https://terasolunaorg.github.io/guideline/1.0.1.RELEASE/en/Overview/SpringMVCOverview.html>.

PARSON, Dale. Using java reflection to automate extension language parsing. *ACM SIGPLAN Notices*, ACM, v. 35, n. 1, p. 67–80, 2000.

PINTO, W. et al. Ferramenta de controle de acesso baseado em gamificação. In: *CBSoft2019 - Sessão de Ferramentas ().* [S.l.: s.n.], 2019

ROBSON, Karen; PLANGGER, Kirk; KIETZMANN, Jan H; MCCARTHY, Ian; PITT, Leyland. Is it all a game? understanding the principles of gamification. *Business Horizons*, Elsevier, v. 58, n. 4, p. 411-420, 2015.

SANDHU, Ravi; FERRAIOLO, David; KUHN, Richard et al. The nist model forrole-based access control: towards a unified standard. In: *ACM workshop on Role-based access control.* [S.l.: s.n.], 2000. v. 2000, p. 1–11.

SANDHU, Ravi S; SAMARATI, Pierangela. Access control: principle and practice. *IEEE communications magazine*, IEEE, v. 32, n. 9, p. 40–48, 1994.

SILVA, Jefferson O; GUERRA, Eduardo Martins; FERNANDES, Clovis Torres. *Esfinge Guardian - Framework para Controle de Acesso*. 2013. Disponível em: <http://esfinge.sourceforge.net/Guardian.html>.

SILVA, Jefferson O; GUERRA, Eduardo M; FERNANDES, Clovis T. An extensibleand decoupled architectural model for authorization frameworks. In: SPRINGER. *International Conference on Computational Science and Its Applications.* [S.l.], 2013. p.614–628.

SKINNER, Burrhus Frederic. *The behavior of organisms: An experimental analysis.* [S.l.]: BF Skinner Foundation, 1990.