UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LUCAS ASTH ASSUNÇÃO
VICTOR PEDRO RODRIGUES LISBOA

UMA PROPOSTA DE MELHORIA DE INTEGRAÇÃO PARA OS SERVIÇOS DO SIGA UFRJ

RIO DE JANEIRO
2022

LUCAS ASTH ASSUNÇÃO
VICTOR PEDRO RODRIGUES LISBOA

UMA PROPOSTA DE MELHORIA DE INTEGRAÇÃO PARA OS SERVIÇOS DO SIGA UFRJ

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Eber Assis Schmitz, D.Sc.

RIO DE JANEIRO
2022

Assunção, Lucas Asth
A851p Uma proposta de melhoria de integração para os serviços do SIGA UFRJ / Lucas Asth Assunção e Victor Pedro Rodrigues Lisboa. – 2022.

86 f.

Orientador: Eber Assis Schmitz.

Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Federal do Rio de Janeiro, Instituto de Computação, Bacharel em Ciência da Computação, 2022.

1. SIGA. 2. Universidade Federal do Rio de Janeiro. 3. REST. I. Lisboa, Victor Pedro Rodrigues. II. Schmitz, Eber Assis (Orient). III. Universidade Federal do Rio de Janeiro, Instituto de Computação. IV. Título.

LUCAS ASTH ASSUNÇÃO
VICTOR PEDRO RODRIGUES LISBOA

UMA PROPOSTA DE MELHORIA DE INTEGRAÇÃO PARA OS SERVIÇOS DO SIGA UFRJ

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 21 de Novembro de 2022

BANCA EXAMINADORA:

Prof. Eber Assis Schmitz, D.Sc

Part. por videoconferência
Prof. Sildenir Alves Ribeiro, D.Sc

Part. por videoconferência
Prof. Claudio Miceli de Farias, D.Sc

Aos membros do SIGA e da comunidade da UFRJ que perpetuam este ambiente de aprendizado.

AGRADECIMENTOS

Lucas Asth Assunção

Agradeço aos meus pais por todo o apoio ao longo dessa jornada, possibilitando que seguisse o curso até o fim.

Meus agradecimentos ao professor Eber Schmitz, por toda paciência e orientação ao longo das revisões deste trabalho. Agradeço também aos professores Claudio Miceli e Sildenir Ribeiro, por suas contribuições para a versão final.

Por último gostaria de agradecer a todos os amigos e companheiros do curso, em especial aos membros do SIGA, pelos momentos de aprendizado.

Victor Pedro Rodrigues Lisboa

Agradeço aos meus pais pelo suporte ao longo do curso. Aos colegas de curso com quem dividimos nossas horas de estudo e trabalho. Agradeço àqueles que participaram da equipe de desenvolvedores do SIGA da UFRJ comigo, por terem partilhado este ambiente e terem proporcionado tanto chance de aprender quanto também de ensinar. Por último, porém não menos importante, gostaria de agradecer aos professores que nos acompanharam durante o curso. Em especial aos nossos orientadores, os professores Eber e Cláudio, pela paciência e orientação durante este trabalho.

"The first principle is that you must not fool yourself."

(Richard P. Feynman)

RESUMO

O objetivo deste trabalho foi buscar melhorias para a capacidade de integração do sistema de gestão acadêmica da UFRJ de forma que estas possam ser absorvidas de maneira sustentável pela equipe responsável por seu desenvolvimento. No primeiro momento observamos características gerais dos sistemas de informação voltados para a gestão de recursos, a fim de obter uma visão geral dos pontos em que podemos focar. Em seguida detalhamos uma sequência de passos para atingir o objetivo de nosso trabalho. Fornecemos também um exemplo seguindo estes passos. A forma que escolhemos para alcançar o objetivo proposto foi através do uso da arquitetura REST, expondo os serviços da aplicação via uma interface que possa ser utilizada, tanto pelos diferentes componentes do SIGA quanto também por outros sistemas da UFRJ ou até mesmo agentes externos à instituição. Usando nosso exemplo de implementação esperamos identificar possíveis problemas que surgiriam de nossa abordagem. Concluímos nosso trabalho delineando os desafios encontrados e aspectos que futuramente possam ser melhorados e adicionados à nossa implementação conforme julgado necessário.

Palavras-chave: SIGA; UFRJ; REST;

ABSTRACT

The goal of this work was to search for improvements related to the integration capacity of UFRJ's academic management system such as they could be taken in a sustainable way by it's development team. First we observed major characteristics of information systems applied to resource management, as a way to obtain an overview of the points we could focus on. We then specify a sequence of steps to achieve the objective of our work. We give an example that follows these steps. The way we chose to achieve the proposed objective was through the use of an architecture called REST, exposing an interface which could be used by the different components from SIGA and possibly by other systems inside UFRJ and even by agents outside the institution. Using our example we hope to identify possible problems that would arise from our approach. At the close of our work we outline the challenges that we met and aspects that could be improved and added to our implementation in the future as judged necessary.

Keywords: SIGA; UFRJ; REST;

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	**A**pplication **P**rogramming **I**nterface
CI	**C**ontinuous **I**ntegration
CRM	**C**ustomer **R**elationship **M**anagement
DAO	**D**ata **A**ccess **O**bject
ERP	**E**nterprise **R**esource **P**lanning
IDE	**I**ntegrated **D**evelopment **E**nvironment
IMPACT	**I**nventory **M**anagement **P**rogram and **C**ontrol **T**echniques
JDK	**J**ava **D**evelopment **K**it
JEE	**J**ava **E**nterprise **E**dition
JSF	**J**ava **S**erver **F**aces
JSON	**J**ava**S**cript **O**bject **N**otation
JVM	**J**ava **V**irtual **M**achine
JWT	**J**SON **W**eb **T**oken
MRP	**M**aterial **R**equirements **P**lanning
MRP II	**M**anufacturing **R**equirements **P**lanning
OID	**O**bject **I**dentifier
ORM	**O**bject-**R**elational **M**apping
REST	**Re**spresentational **S**tate **T**ransfer
SaaS	**S**oftware **a**s **a** **S**ervice
SCM	**S**upply **C**hain **M**anagement
SGA	**S**istema de **G**estão de **A**cesso
SIGA	**S**istema **I**ntegrado de **G**estão **A**cadêmica
SOA	**S**ervice-**O**riented **A**rchitecture
TDD	**T**est **D**riven **D**evelopment

TI	**T**ecnologia da **I**nformação
UFRJ	**U**niversidade **F**ederal do **R**io de **J**aneiro
XML	E**x**tensible **M**arkup **L**anguage

LISTA DE SÍMBOLOS

kB	Quilobyte
MB	Megabyte
GB	Gigabyte

SUMÁRIO

1 INTRODUÇÃO

Ao abordar sistemas de informação entre seus elementos mais marcantes podemos incluir sua capacidade de processar e interpretar dados (NIST, 2022). Mesmo sendo capazes de executar tais tarefas com precisão o acesso a estas funcionalidades é de suma importância. Desta forma é necessário que tais artefatos possuam a capacidade de ofertar a seus clientes formas de acesso que sejam convenientes.

1.1 SISTEMAS DE INFORMAÇÃO NA UFRJ

O advento dos sistemas de informação impulsionou nas últimas décadas uma transformação no que tange a precisão e celeridade no tratamento de dados, disposição de informações e execução de processos em várias áreas de atuação.
Dentre as diversas instituições que fazem uso de elementos como estes podemos listar a Universidade Federal do Rio de Janeiro (UFRJ). Sendo concebida em 1920 (FÁVERO, 1999) ela é uma das principais instituições de ensino superior do Brasil, com um corpo discente composto por dezenas de milhares de alunos. As atividades diárias, tanto acadêmicas quanto administrativas, geram um montante considerável de dados sendo estes processados e armazenados por diversos sistemas, dentre estes estão os que compõe o Sistema Integrado de Gestão Acadêmica (SIGA) da UFRJ.
Como objeto de nosso estudo iremos focar em um dos principais componentes do SIGA, o Portal do Aluno. Apesar desta restrição podemos argumentar que as similaridades entre os seus demais elementos serão suficientes para aplicação de nossos resultados.

1.2 OBJETIVOS DA PESQUISA

Neste trabalho propomos uma forma alternativa de expor os dados e serviços do SIGA. Queremos facilitar seu consumo e possibilitar sua integração com outras aplicações. O principal ganho que esperamos obter é a capacidade de servir tais funcionalidades além da forma atual que é através de páginas *web*.
Desejamos obter resultados que sejam aplicáveis de maneira homogênea entre os artefatos que compõe o SIGA. Assim determinamos um método para que possamos, de maneira consistente, identificar e servir seus dados e serviços utilizando a sua estrutura assim como padrões de projeto. Através deles é que tornaremos possível que suas funcionalidades sejam consultadas tanto internamente pelos componentes do SIGA quanto por agentes externos.
A fim de prover uma abordagem mais detalhada em nosso trabalho nos limitamos a um dos sistemas que compõe o SIGA, o Portal do Aluno. Restringindo nosso escopo

e escolhendo cuidadosamente as ferramentas usadas no exemplo de implementação que criamos esperamos tornar nosso trabalho compatível com a forma de desenvolvimento atual da equipe. O que amplia a aplicabilidade deste trabalho ao cenário em que o SIGA está inserido.

1.3 ESTRUTURA DO TRABALHO

No primeiro momento de nosso trabalho argumentamos a relevância do objetivo escolhido. No capítulo 2 começamos fazendo um breve resumo da evolução dos *softwares* voltados para gestão de recursos. No capítulo 3 apresentamos características do Portal do Aluno[1] e as relacionamos com as observações feitas sobre os *softwares* de gestão. Apontamos casos onde ampliar a capacidade de integração do SIGA seria proveitoso, levando em conta tanto componentes internos quanto agentes externos ao próprio SIGA. Detalhamos brevemente a equipe responsável pelos artefatos que abordamos.
Partindo destes detalhes determinamos uma sequência de passos para levar os componentes do Portal de sua organização atual para outra arquitetura. Esta servirá como meio para atingir o objetivo que estipulamos. Isto compõe o capítulo 4 deste trabalho.
No capítulo 5 exemplificamos o uso do método proposto. Escolhemos uma parte do Portal do Aluno para aplicarmos os passos que determinamos. Os detalhes que fornecemos anteriormente sobre o Portal servem para restringir as escolhas que adotamos nesta implementação.
Ao final ressaltamos resultados, tanto favoráveis quanto indesejáveis, fruto de nossas escolhas no capítulo 6. Concluímos no capítulo 7 pontuando formas com que nosso trabalho possa ser continuado, contemplando funcionalidades do Portal do Aluno que não tocamos em nossa implementação.

[1] https://portal.ufrj.br

2 ERP E *SOFTWARES* DE GESTÃO

Neste capítulo consultamos alguns autores sobre as diferentes etapas na evolução das aplicações relacionadas com a gestão de recursos.
Através de suas diversas iterações até os dias atuais elas se tornaram capazes de se adaptar a diversos domínios de atuação. Em seus desenvolvimentos mais recentes sendo capazes de interagir com diversos agentes, até mesmo externos à aplicação. Um dos acontecimentos mais recentes neste sentido que exemplifica bem isto são os *softwares* de Planejamento de Recursos Empresariais, também conhecidos como *Enterprise Resource Planning* ou ERP.

2.1 OS ERP E OUTROS *SOFTWARES* DE GESTÃO

2.1.1 Primeiros passos

O surgimento dos ERP não ocorreu de maneira súbita, sendo um resultado que agrega décadas de transformações e avanços na forma e disponibilidade de recursos computacionais. Estas mudanças possibilitaram gradativamente que diferentes setores das organizações que antes não faziam uso da Tecnologia da Informação (TI) fossem contemplados por novos sistemas.
Segundo (WORTMANN, 1998), um dos primeiros passos neste sentido foi dado nos anos 60 com artefatos como o *IBM IMPACT*. O foco neste caso era o controle de inventário para um grande número de itens, provendo otimizações baseadas em previsões de demandas futuras. Kleijenen e Rens fornecem detalhes mais específicos sobre o funcionamento deste pacote (RENS; KLEIJNEN, 1978).
Em 1962, quando o *IMPACT* foi lançado os computadores eram, em determinados aspectos, dissimilares aos que possuímos hoje. A diferente forma de entrada de dados e as limitações em capacidade de processamento se traduziam em usos específicos, tornando sua aplicação isolada dos demais processos dentro das organizações que os utilizavam. Neste caso específico que citamos estando limitado ao controle de inventário e alimentado manualmente através de fitas magnéticas e cartões perfurados, conforme era o padrão de entrada de dados da época. Exemplos de máquinas destes tipo podem ser encontrados no *PDP-1* (BELL, 1961) e no *IBM 1440* (IBM, 2020).
Ao final dos anos 60 surgem exemplos de *softwares* para suporte ao chamado Planejamento de Necessidades Materiais (do inglês *Material Requirements Planning* ou MRP). Nele diferentes etapas da produção precisam ser levadas em conta pelo programa de forma que a quantidade de recursos físicos e tempo necessários possam ser estimados (WORTMANN, 1998). Uma descrição detalhada dos sistemas MRP não é um dos objetivos de nosso trabalho, aqui o enxergamos apenas como uma etapa na sequência de inovações que levariam ao desenvolvimento dos ERP. Maiores detalhes sobre este modelo de pla-

nejamento de produção podem ser consultados em (LAURINDO; MESQUITA, 2000). Podemos apontar que tanto no caso do *IMPACT* quanto do MRP o usuário final (quem consome os produtos de fato) ainda não interagia com a aplicação. Apesar disso neste salto entre estes *softwares* podemos observar um incremento na quantidade de fluxos e atividades da organização que começam a ser considerados pela aplicação. Passando a englobar não apenas o planejamento de estoque mas agora também o envolvimento de materiais na produção.
Nos anos subsequentes novos módulos são adicionados aos MRP, encompassando outros setores além de planejamento e produção. A execução das rotinas destes sistemas passaram a acontecer em computadores com tempo compartilhado (do inglês *time-sharing*) ao invés daqueles que tinham como base o processamento em lote (ou *batch*) (MCGAUGHEY; GUNASEKARAN, 2007). Para mais detalhes sobre a mudança de paradigma dos sistemas em *batch* para tempo compartilhado consulte (SACKMAN, 1968). Wortmann atribui esta integração de diferentes setores principalmente a duas inovações que ocorreram nos anos 70: o processamento *online* e os pacotes para gerenciamento de bancos de dados. Para ele o primeiro aspecto reflete uma melhoria no carregamento dos dados usados pelo sistema que agora poderia ser feito via terminal. O segundo aspecto permite que os sistemas MRP abordem mais áreas dentro da empresa como finanças e logística (WORTMANN, 1998).

Entre as décadas de 70 e 80 o MRP teve sua continuação com o chamado Planejamento de Recursos de Produção (do inglês *Manufacturing Resources Planning* ou MRP II). Enquanto no MRP o foco é restrito na gerência de materiais e insumos, os sistemas MRP II abordam uma parcela maior dos componentes e fluxos empresariais, não estando restritos apenas aos insumos utilizados no processo de produção. Podemos listar mecanismos voltados para o controle de qualidade, contabilidade e *marketing* como exemplos de funcionalidades adicionais que diferenciam o MRP do MRP II. Uma abordagem mais detalhada do MRP II pode ser encontrada em (WIGHT, 1995). Wortmann ressalta que do ponto de vista da TI a importância desta iteração no desenvolvimento dos pacotes de *software* se dá no sentido em que surgem evidências de que artefatos padronizados poderiam ser desenvolvidos, servindo diferentes áreas de interesse (WORTMANN, 1998). Esta reutilização se tornaria um padrão característico dos ERP, que possuem a capacidade de serem ajustados a diferentes domínios. Outro ponto importante para este autor é a mudança no ambiente onde estes programas eram executados. Mudando dos *mainframes* para arquiteturas cliente-servidor, em especial para aquelas que faziam uso de código aberto como por exemplo sistemas operacionais baseados no UNIX. Este padrão se mantém até hoje, uma vez que muitas empresas utilizam *software* aberto e também contribuem para eles seguindo o modelo *open-source* [1]. Autores como McGaughey e Gunasekaran ressaltam que o MRP II também ficou marcado pela substituição dos sistemas

[1] <https://www.redhat.com/en/topics/open-source/what-is-open-source>

de arquivos por bancos de dados e o uso da rede para comunicação entre os componentes dos sistemas MRP II. Segundo eles estes componentes se encontravam, em alguns casos, separados geograficamente (MCGAUGHEY; GUNASEKARAN, 2007).

2.1.2 Os ERP

Ao final dos anos 80 e ínicio dos 90 surge o *Enterprise Resource Planning*, este tipo de *software* ficaria conhecido pela abrangência de sua capacidade de integrar os diversos elementos que compõe um negócio (RASHID; HOSSAIN; PATRICK, 2002). Desta maneira cobrindo grande parte dos processos empresariais e fornecendo uma visão unificada deles, onde cada setor é atendido por um dos módulos do programa. Olhando para as etapas anteriores da evolução dos *softwares* voltados para gestão podemos perceber que este incremento da capacidade de integração já era tendência, ocorrendo de certa forma no MRP e MRP II porém agora em uma escala ainda mais ampla. Estes *softwares* também ficariam marcados pela dificuldade de implantação, devido à necessidade de alinhar as especificidades dos processos da empresa com as particularidades do ERP escolhido e vice-versa.

Com o passar dos anos estes sistemas passariam por mais transformações, provendo um conjunto maior de funcionalidades. Um exemplo apontado por McGaughey e Gunasekaran é o chamado *m-commerce*, ou seja, o comércio de bens e serviços intermediado por dispositivos móveis (*smartphones* em sua maior parte), sendo esperado que os ERP disponham de capacidades adicionais que forneçam suporte para novas plataformas como estas (MCGAUGHEY; GUNASEKARAN, 2007).

Por volta dos anos 2000 esta e outras mudanças transformaram o conceito inicial dos ERP a ponto de alguns autores criarem uma nova nomenclatura para este novo tipo de produto, o chamado ERP II. Estes autores apontam também que novos modelos de negócio envolvendo a interação de múltiplas empresas refletia uma intensificação da necessidade de que estes artefatos fossem capazes de possibilitar que os processos entre empresas distintas se comunicassem(MCGAUGHEY; GUNASEKARAN, 2007). Algo que expandiria a definição dos ERP da forma como foram originalmente envisionados, reforçando a necessidade desta nova nomenclatura para estes sistemas. Autores como Hurbean e Fortache listam, além de capacidades relacionadas ao fluxo interno das empresas, módulos oferecendo funcionalidades que abrangem os elementos externos ao negócio como *Supply Chain Management (SCM)*, *Customer Relationship Management (CRM)* e *Data Warehousing (DW)*, extendendo o sistema para além da empresa (HURBEAN; FOTACHE, 2014).

No DW podemos levar em conta fontes externas ao negócio para contribuir para a geração de relatórios, no SCM podemos ter fornecedores que são agentes externos à empresa e no CRM é possível utilizar insumos externos, como redes sociais por exemplo, para conhecer

melhor os clientes em questão. Uma explicação mais detalhada destas tecnologias pode ser encontrada na *web*[2][3][4].

Na iteração mais recente dos ERP (de 2010 até os dias atuais), listada pelos autores que consultamos, a mudança mais expressiva foi a escolha pelo ambiente de computação em nuvem para abrigá-los. Neste os sistemas são utilizados pelos clientes seguindo o modelo de *software* como um serviço (do inglês *Software-as-a-Service* ou SaaS). Nele o fornecedor do ERP é responsável por manter as instâncias do sistema em execução, que são acessadas por quem contrata este serviço e por seus clientes via uma interface *web*. Do que nos interessa dentro do escopo deste estudo que é a visão do mantenedor destes artefatos um elemento sobressai, o consumo de recursos. Dos autores que consultamos previamente Elmonen, Nasr e Geith ressaltam também em sua discussão sobre o *Cloud ERP* a questão da segurança da informação. Segundo eles a maior exposição do sistema via rede fez com que ele herdasse as preocupações relacionadas à segurança inerentes ao modelo de *cloud computing* (RASHID; HOSSAIN; PATRICK, 2002). Apesar de tal preocupação ser válida não abordaremos esta parte voltada para a segurança da informação neste trabalho.

Tabela 1 – Sumário das transformações ressaltadas

Período	Transformação
1960	Gerência de inventário
1970	MRP
1980	MRP II
1990	ERP
2000	ERP II
2010	Cloud ERP

Fonte: tabela do autor

2.1.3 Pontos de interesse

Das características que reunimos acima podemos ressaltar duas que consideramos de maior interesse se tratando dos *softwares* voltados para gestão e que estão presentes nos ERP. Primeiro a sua presença em diversas áreas de um negócio onde cada uma de suas partes atua em um aspecto diferente. No cenário do SIGA isto se traduz em haver diversos componentes voltados para os diferentes aspectos da gestão acadêmica. Estes componentes devem então estar integrados de maneira que os dados dos processos acadêmicos sejam consultados pelos diferentes componentes do sistema assim que necessário e também pelo usuário final. Os dados também podem ser agregados fornecendo uma visão geral do funcionamento da instituição para os usuários de caráter administrativo. Em segundo lugar,

[2] <https://aws.amazon.com/data-warehouse/>
[3] <https://www.ibm.com/topics/supply-chain-management>
[4] <https://www.salesforce.com/crm/what-is-crm/>

devido ao fato de os sistemas que citamos acima, com exceção do SIGA, serem produtos é interessante ao vendedor que eles possibilitem sua inserção em cenários distintos. É neste ponto que entra a importância da capacidade de customização destes produtos. Para caracterizar o SIGA como um ERP por exemplo, esta capacidade de customização seria imprescindível. Além destes dois pontos as iterações mais recentes dos ERP incluem também outras características interessantes. Citamos o envolvimento do comércio eletrônico e uso através de *smartphones*. Isto demonstra que atualmente a integração dos componentes constituintes dos ERP também precisam levar em conta agora clientes e agentes externos com um maior grau de heterogeneidade. Visto que o acesso passa agora a ser feito a partir de diferentes tipos de dispositivos.

Ressaltamos a mudança de ambiente escolhido para abrigar as aplicações que listamos, evidente no desenrolar do *cloud* ERP. Isto evidencia que o uso de recursos feito de forma mais competente também é um fator a ser considerado. Uma vez que em um ambiente de computação em nuvem o consumo de recursos passa a ser taxado com uma granularidade mais fina. Na próxima seção abordaremos o SIGA da UFRJ. Fornecemos uma visão geral sobre ele e o relacionamos com as características observadas nos ERP e seus antecessores. Com esta comparação esperamos justificar as melhorias que propomos aplicar ao SIGA, que é facilitar sua integração com outros elementos, expondo seus serviços de maneira diferente da forma atual. Detalhar a realidade na qual o SIGA está inserido tornará possível uma análise das restrições relativas ao custo de implementação de melhorias. O objetivo de identificar e respeitar tais restrições ajudará a manter a aplicabilidade do nosso trabalho.

3 DEFINIÇÃO DOS OBJETIVOS DA PESQUISA

Neste capítulo detalhamos o objetivo que escolhemos para o nosso trabalho e sua importância para os artefatos envolvidos.
Ele amplia a capacidade de integração do SIGA, possibilitando que os seus serviços possam ser utilizados tanto pelos próprios componentes dos sistemas que o compõe quanto por agentes externos.

3.1 O SIGA UFRJ

Usando o que foi exposto anteriormente podemos argumentar similaridades de alguns componentes do SIGA com certas características que os tipos de aplicação que lá mencionamos possuem. Para isso precisamos apresentar antes estes componentes, falaremos deles a seguir explicando como eles estão organizados. Começamos com uma breve descrição da equipe que os mantém e em seguida detalhamos os elementos específicos do SIGA que iremos abordar.

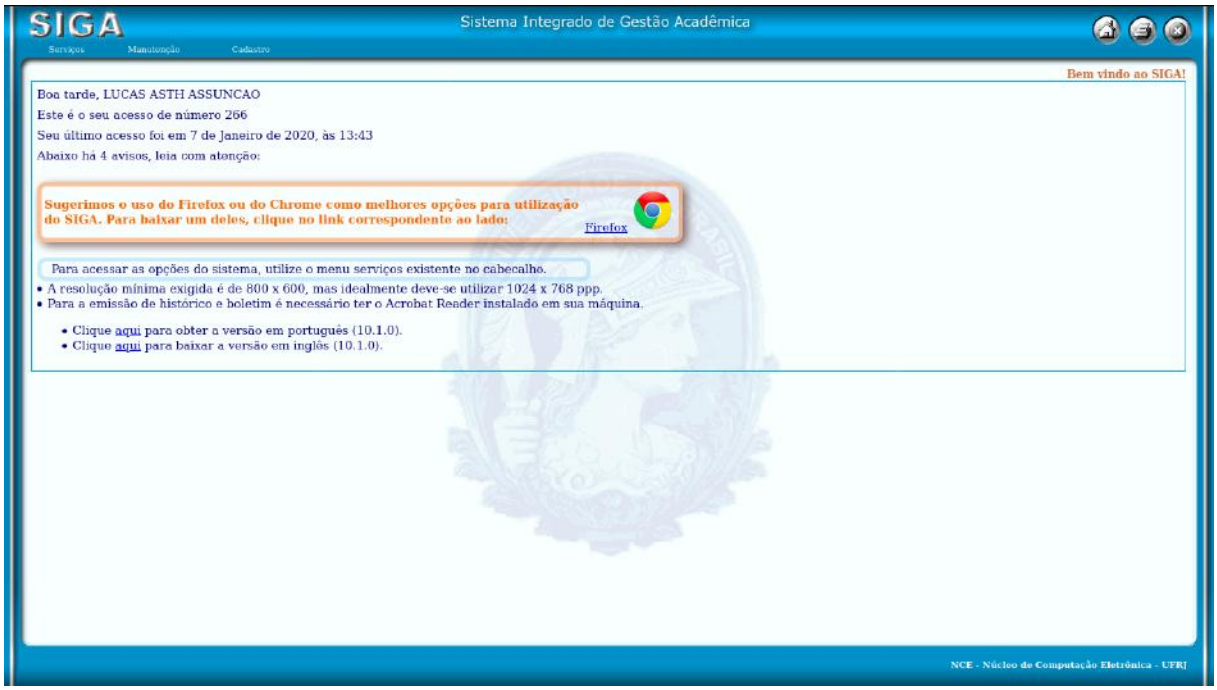
3.1.1 A equipe responsável pelo SIGA UFRJ

A equipe que mantém e desenvolve o SIGA conta com membros que são em sua maioria alunos ou ex-alunos dos cursos de graduação da UFRJ. Os desenvolvedores são principalmente dos cursos de Ciência da Computação e Engenharia Eletrônica e de Computação. Dentre os artefatos sob responsabilidade da equipe podemos ressaltar 4 elementos: o Sistema de Gestão de Acesso (SGA), o sistema Pré-Matricula, a versão legada do sistema de gestão acadêmica (figura 1) que atualmente tem acesso restrito aos técnicos administrativos e finalmente o Portal do Aluno (figura 2). Existem também as versões *mobile* de alguns dos serviços do Portal do Aluno, disponíveis na forma de aplicativos para *Android*[1] e *iOS*[2].
Dado o tamanho da comunidade atendida pelo SIGA é natural que os períodos de ociosidade da equipe com relação a demandas sejam raros. O que nos leva a primeira restrição: qualquer tentativa de implantação de melhorias nestes sistemas está restrita pelas necessidades e atendimento da comunidade acadêmica. Uma vez que esta demanda imposta pela comunidade da UFRJ sobre a equipe é considerável, partimos do princípio que uma nova visão, qualquer que ela seja, sobre a implementação de funcionalidades deve levar em conta o custo com o qual a equipe teria de arcar. Enfatizando o lado técnico deste custo iremos detalhar na próxima seção as ferramentas escolhidas para a implementação que criamos. Com isto esperamos exemplificar a compatibilidade com a forma de desen-

[1] <https://www.android.com/>
[2] <https://www.apple.com/ios>

volvimento que já existe atualmente no SIGA. Esperamos que estas semelhanças sejam capazes de prover um baixo custo de manutenção e extensão.
É desejável que exista um elevado grau de compatibilidade entre as abordagens, que propomos e a já existente, aos problemas endereçados pelo SIGA. Visto que em um cenário de migração estas duas abordagens irão coexistir durante um período de tempo. Para detalhes sobre o SIGA que estão além do viés técnico e sobre o uso da TI na UFRJ em geral consulte (STORINO, 2017).

Figura 1 – Tela inicial da versão anterior ao Portal do Aluno

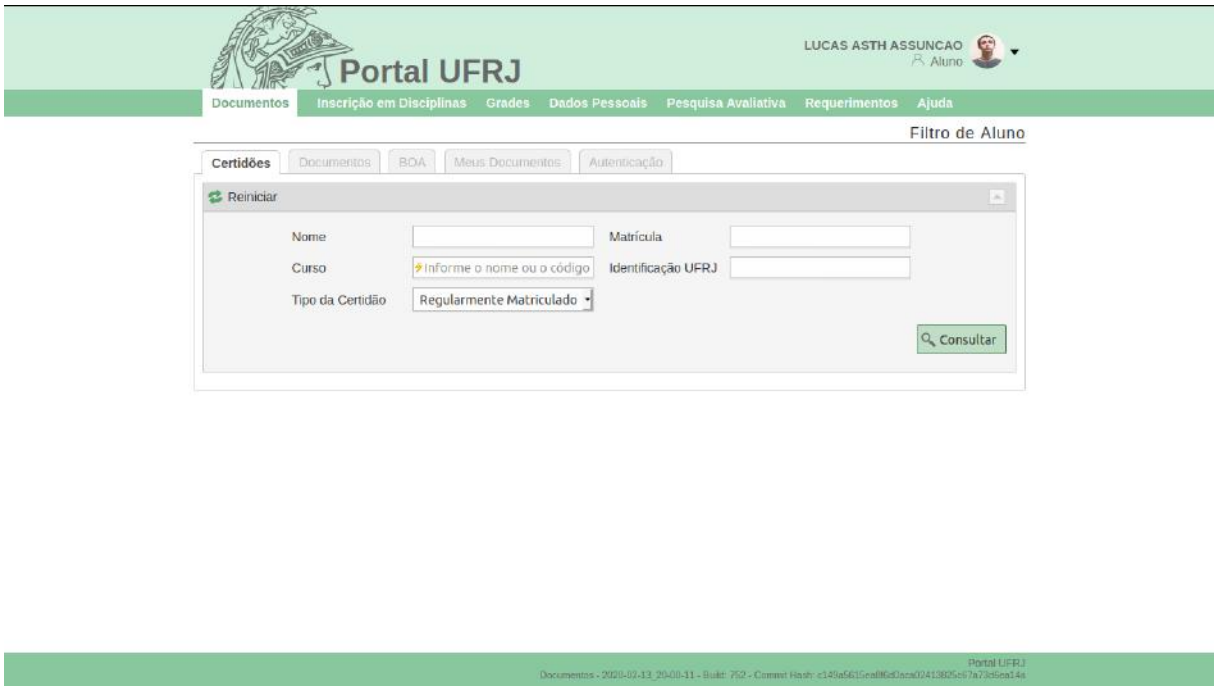


Fonte: figura do autor

3.1.2 O Portal do Aluno

O SIGA é composto por diferentes partes mas aqui iremos abordar apenas uma delas, o Portal do Aluno. Ele é um conjunto de aplicações *web* feitas em *Java*. O Portal é dividido em diversos módulos, onde cada um destes é um projeto separado que agrega serviços voltados para uma determinada área. Atualmente a maior parte dos serviços do SIGA em uso está concentrada nos módulos do Portal. A emissão de documentos fica a cargo do módulo Documentos e a inscrição em disciplinas é responsabilidade do módulo Inscrição por exemplo. Outros exemplos podem ser encontrados na tabela 1.
Estes projetos são organizados seguindo o padrão *Model-View-Controller* (MVC). Nele é feita a separação dos componentes da aplicação em três partes que juntas dão o nome ao padrão. A camada de visualização (*view*) contém a interface utilizada pelo usuário para interagir com a aplicação (como por exemplo na figura 2). O modelo (*model*) contém a

Figura 2 – Tela do módulo Documentos do Portal do Aluno



Fonte: figura do autor

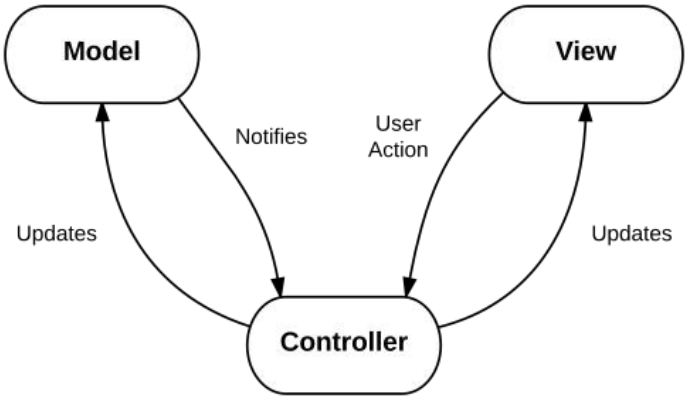
lógica da aplicação referente a regras de negócio e acesso ao banco de dados. Através dele é que são criados, recuperados e atualizados os dados da aplicação. O controlador (*controller*) recebe os dados das ações do usuário e os repassa para o modelo, ele contém as ações que podem ser executadas pelo usuário na camada de visualização. Ações como por exemplo a navegação entre as telas da aplicação e o envio de dados de um formulário. Mais detalhes sobre o padrão de projeto MVC podem ser encontrados em (MAJEED; RAUF, 2018). Na figura 3 se encontram os elementos do padrão MVC.

Citamos nas seções anteriores como os módulos dos ERP podem depender de outros módulos e suprir dados para eles. Em suas iterações mais recentes podendo fornecer estes dados até mesmo para elementos externos ao próprio sistema. No Portal do Aluno um cenário similar acontece porém, atualmente, restrito aos seus próprios componentes. No cenário atual existe a necessidade de um módulo obter dados que competem a outro módulo do Portal. Ela é endereçada da seguinte forma: as classes que compõe o modelo do módulo desejado (como exemplificado na figura 4) são empacotadas em um arquivo no formato JAR (*Java Archive*[3]). Assim os módulos que necessitarem dos serviços de outro módulo precisam importar o seu respectivo arquivo JAR. Estes arquivos são versionados em um repositório mantido pela própria equipe. Desta forma estas dependências (assim como suas versões específicas) tem de ser declaradas nos projetos que delas dependem. Em nossa proposta iremos fornecer um mecanismo para endereçar estas dependências de

[3] <https://docs.oracle.com/javase/6/docs/technotes/guides/jar/index.html>

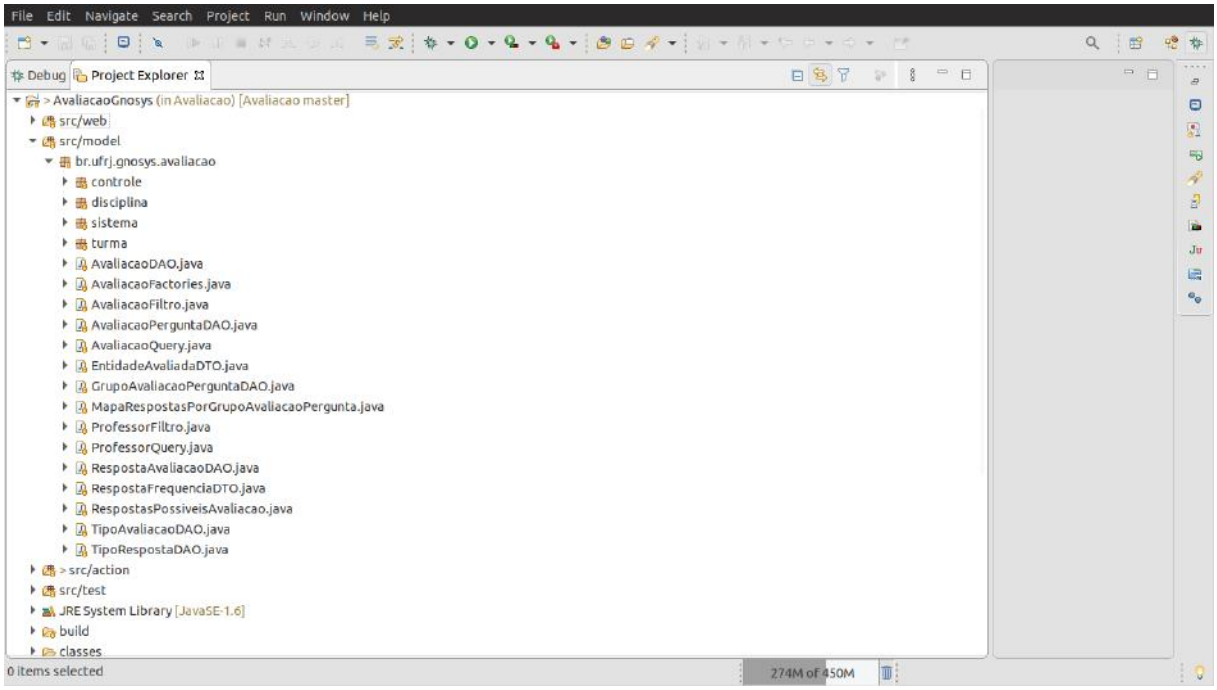
outra maneira.

Figura 3 – Elementos do padrão MVC



Fonte: Wikimedia Commons (2017)

Figura 4 – Modelo do projeto Avaliação no ambiente de desenvolvimento



Fonte: figura do autor

3.2 OBJETIVOS ESCOLHIDOS

Das duas capacidades que nos chamaram atenção ao observar os *softwares* voltados para gestão focaremos em apenas uma para guiar nosso trabalho. Esta seria a possibilidade destes artefatos serem integrados com outros elementos. Acreditamos que o SIGA

Tabela 2 – Alguns dos módulos do Portal do Aluno

Portal do Aluno
Avaliação
Documentos
Registro
Ensino
Segurança
Inscrição
Histórico
Bolsa
Comunicação
Turma
Indicadores

Fonte: tabela do autor

possa alcançar um novo patamar no que diz respeito a este quesito. Podemos ressaltar dois exemplos que ilustram o espaço existente para esta mudança. Primeiro podemos citar os aplicativos *mobile* feitos pela equipe. Estes precisam ter acesso aos dados e serviços do Portal para funcionar, porém da forma como o Portal foi pensado apenas a interface *web* acessada via navegador (figura 2) estava disponível a princípio. Sendo assim as funcionalidades das quais eles precisam foram servidas de forma alternativa, uma vez que estes aplicativos compõe projetos separados e não fazem parte do Portal. Em seguida podemos citar o projeto Caronaê[4], cujo o aplicativo necessitava consultar dados de alunos presentes no Portal, a fim de ajudar a identificar se os participantes da plataforma constituiam parte do corpo discente da UFRJ. Mais uma vez precisando criar algo fora do padrão do Portal do Aluno para atender a este caso. Em nossa proposta queremos que esta capacidade esteja presente por padrão. De forma que novas ocorrências que precisarem de outros dados ou serviços não necessitem ser tratadas caso a caso. No próximo capítulo organizaremos um método para chegarmos a uma nova organização do Portal do Aluno com o intuito de atingir este objetivo.

[4] <https://caronae.org/>

4 MÉTODO PROPOSTO

Neste capítulo detalhamos passos para alcançar o objetivo especificado no capítulo anterior, em seguida produzimos um exemplo seguindo eles. Descrevemos também as ferramentas e arcabouços que usamos, comparando-as com as que são utilizadas no Portal do Aluno atualmente. Esperamos deixar evidente a motivação por trás da escolha de substitutos para alguns dos arcabouços originalmente usados no Portal. Assim como a escolha por manter alguns daqueles que já eram utilizados.
Uma divisão de conceitos comum feita em aplicações *web* é a separação entre os chamados *front-end* e *back-end*. Nesta primeira parte ficam os componentes responsáveis pela apresentação dos dados consultados e o recebimento da entrada fornecida pelos usuários para a aplicação. Podemos citar as interfaces gráficas com as quais o usuário interage como exemplo. Na segunda parte constam os componentes da aplicação que são capazes tanto de processar os dados quanto armazená-los ou recuperá-los de onde eles foram persistidos. Em nosso trabalho focamos apenas nesta segunda parte.

4.1 IDENTIFICANDO SERVIÇOS

Nossa proposta se resume em ampliar o acesso aos dados e diferentes serviços que compõe o Portal do Aluno, os expondo de uma maneira distinta da qual é feita atualmente. Nosso primeiro passo é identificar onde estes elementos estão localizados na aplicação em questão. Nos módulos do Portal, por estarem organizados seguindo o padrão MVC, podemos começar observando sua camada de modelo. Nesta parte do projeto encontraremos as classes responsáveis pela lógica executada em seus serviços, assim como pelo acesso e persistência dos dados. Podemos observar também os testes automatizados da aplicação para auxiliar nesta etapa. Um dos propósitos dos testes é certificar que determinados casos de uso estão sendo atendidos. Assim eles capturam algumas das maneiras como as funcionalidades do sistema podem ser utilizadas. Desta forma as classes de teste do projeto também podem ajudar a identificar quais serviços estão presentes em um determinado módulo. Estes arquivos podem ser encontrados no caminho *src/test* de cada módulo, conforme o exemplo da figura 4.
Na camada de modelo de cada um dos módulos do Portal do Aluno podemos identificar quais entidades estão em uso por ele. Para tal podemos focar em uma abstração que é utilizada para a busca e persistência de dados. Esta abstração é o padrão de desenvolvimento chamado *Data Access Object* (Objeto de Acesso a Dados) ou DAO. Seu objetivo é abstrair detalhes dos mecanismos de persistência da aplicação para outras classes do modelo que dependem destas capacidades[1]. Uma vantagem deste padrão é possibilitar que

[1] <https://www.oracle.com/java/technologies/data-access-object.html>

mudanças na implementação da camada de persistência possam ocorrer tendo pouca ou nenhuma influência sobre as funcionalidades da aplicação que dependem dela. Nas classes dos DAO encontramos os métodos que representam operações básicas como buscar, criar, remover e atualizar entidades do modelo da aplicação. Tais entidades são definidas como classes que representam os elementos do domínio ao qual o Portal está inserido. Algumas destas entidades são: alunos, cursos, disciplinas e pedidos de inscrição em disciplinas. Um exemplo de um DAO do Portal do Aluno pode ser encontrado no anexo D.1. Um ponto relevante para nós é o fato de quase toda entidade do modelo possuir um DAO associado. Podemos assim descobrir a maior parte das entidades que compõe o projeto realizando uma busca por arquivos que tenham o nome contendo o termo *DAO.java*. Esta busca pode ser feita utilizando o ambiente de desenvolvimento integrado (IDE) adotado pela equipe, exemplificado na figura 4.

Tendo uma forma sistemática de identificar as entidades envolvidas em um dado módulo precisamos agora de uma maneira de listar os seus serviços. Estes podem ser encontrados em seus diferentes pacotes e pastas dentro do caminho *src/model*. Cada qual contendo os arquivos referentes a um serviço específico. Nos casos onde esta organização não é seguida rigorosamente podemos consultar o arquivo *pages.xml* do módulo em questão. Neste arquivo é feito o mapeamento entre as interfaces que serão renderizadas pelo navegador do usuário (*web browser*) e os métodos de seus controladores associados, que no caso são classes *Java*. Na definição destes controladores podemos encontrar os métodos que representam ações executadas pelos usuários ao interagir com uma determinada página. Estas ações irão envolver classes do modelo da aplicação que contém os serviços que queremos identificar neste momento. A principal forma que os usuários interagem com os serviços do Portal atualmente é através destas páginas que estão cadastradas no arquivo *pages.xml*. Sendo assim uma busca pelas páginas declaradas neste arquivo irá mostrar todos os serviços associados a um dado módulo que podem ser acessados pelo navegador. Desta maneira temos em mãos um método para identificar também todos os serviços de um dado módulo do Portal do Aluno.

4.2 EXPONDO SERVIÇOS

Dado que temos um meio de identificar os serviços e suas entidades associadas podemos partir para a próxima etapa. Nela vamos determinar passos a serem seguidos para que estes elementos que identificamos possam ser acessados por meios que vão além da forma atual, que é restrita aos navegadores. Para alcançar este objetivo escolhemos adotar elementos da arquitetura *Respresentational State Transfer* (REST). Não abordaremos de maneira exaustiva seus detalhes porém estes podem ser consultados na obra que a define (FIELDING, 2000). Um de seus temas centrais é o conceito de recurso e como estes recursos são manipulados usando interfaces e representações uniformes. No nosso caso um

recurso pode representar tanto uma entidade (como por exemplo um aluno ou um boletim) quanto a situação atual ou resultado da execução de um serviço (como o fechamento de um período letivo). De maneira geral estes serão os dois tipos de recurso que teremos, dados puros e metadados referentes aos serviços que operam sobre estes dados.

Conforme citamos acima uma restrição importante imposta por esta arquitetura é a uniformidade do acesso ao conjunto de recursos que compõe nossa aplicação. Para garantir o acesso desta forma precisamos determinar a estrutura dos identificadores que usaremos para distinguir e acessar nossos recursos. Através destes identificadores conseguiremos realizar as operações que desejamos tais como criar, editar ou remover um recurso. No caso das entidades do modelo da aplicação usamos o identificador único (ou ID) presente no banco de dados. No caso dos serviços utilizamos o nome ou descrição da ação por ele executada para definir o seu identificador. Determinamos a seguinte convenção para estruturar o caminho que identifica os recursos: nome do módulo do Portal ao qual ele está associado, seguido pela entidade e ou serviço que se deseja acessar e finalmente o seu número identificador. Podemos ilustrar melhor com um exemplo, para buscarmos uma inscrição em disciplinas poderíamos fazer uma requisição para *https://portal.ufrj.br/api/v1/Inscricao/inscricao/1*. Neste exemplo 1 é o identificador da entidade inscrição que é responsabilidade do módulo Inscrição. O prefixo */api/v1* serve para versionar e indicar os caminhos que compõe a interface contendo os pontos de acesso para o *backend* da aplicação. Este conjunto de pontos de acesso ou *endpoints* compõe o que é comumente chamado de *Application Programming Interface* ou API. Através deste prefixo deixamos livres as opções de caminhos onde serão requisitadas as páginas do *frontend* da aplicação, que neste exemplo poderia estar localizada em *https://portal.ufrj.br/Inscricao/inscricao* por exemplo. Assim evitamos possíveis conflitos que podem acontecer ao se escolher nomes iguais para estas duas partes distintas.

Usamos o protocolo HTTP para interagir com a aplicação. A forma como especificamos as ações realizadas sobre os recursos é através dos métodos (também chamados verbos) HTTP[2]. Sendo assim para buscar um recurso utilizamos o método *GET*, para criar um novo recurso usamos o *POST*, para atualizar um recurso existente *PATCH* ou *PUT* e para remover um recurso usamos o verbo *DELETE*.

Para que os dados possam ser consumidos por diversos clientes além dos navegadores precisamos que eles estejam em um formato que seja compreendido por todas as partes envolvidas. Escolhemos o formato JSON[3] para representar nossos dados. Principalmente devido a sua ampla adoção atualmente no âmbito do desenvolvimento *web*. Ao receber uma representação de um recurso neste formato o cliente tem em mãos, através deste objeto JSON, as informações necessárias que devem ser renderizadas ao usuário.

[2] <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods>

[3] <https://www.json.org/json-en.html>

4.3 COMPARANDO NOSSA SOLUÇÃO COM A ARQUITETURA ORIENTADA A SERVIÇOS (SOA)

Mencionamos como os projetos que compõem o Portal do Aluno importam, de outros módulos do próprio Portal, algumas das partes que necessitam para funcionar. Devido a isso o projeto acaba por ter um viés que é mais predominante em sistemas monolíticos, mesmo possuindo uma organização modular. Argumentamos isto pois o código da funcionalidade importada reside também de fato no próprio módulo que dela precisa uma vez importado. A forma que propomos para tratar esta necessidade seria consultando o ponto de acesso relacionado ao serviço ou dado que for necessário. Este seria exposto pelo módulo de onde originalmente importaríamos tais funcionalidades. Não haveria necessidade nesse caso de se importar partes do código deste outro módulo durante a etapa de construção do artefato a ser executado. Esta organização que propomos se assemelha ao padrão de arquitetura orientada a serviços ou SOA (*Service Oriented Architecture*). Apesar disso não podemos considerar que nossa implementação adere totalmente a este padrão de projeto. Isto se deve a não termos um barramento comum a todos os serviços por onde eles podem ser acessados. Não temos também um componente onde podemos registrar os serviços para que possam ser descobertos de forma automatizada.
Existem três blocos principais no padrão SOA. O provedor do serviço, que oferece uma interface pela qual o serviço será consumido e se cadastra no repositório de serviços. O repositório de serviços é um outro bloco separado, onde são guardadas as informações sobre como e onde acessar um determinado serviço. Por fim o consumidor do serviço é o elemento que irá consultar o repositório e em seguida consumir o serviço do qual ele precisa. Podemos fazer um paralelo com o nosso cenário usando a implementação que criamos. Nela cada módulo do Portal cumpre o papel de consumidor e provedor de serviços. Estes e outros detalhes do padrão SOA podem ser encontrados em (ZHANG; ZHANG; CAI, 2007).

5 EXEMPLO DE IMPLEMENTAÇÃO

A fim de colocar em prática o que descrevemos acima e ajudar a identificar pontos a serem melhorados criamos um exemplo implementando os passos que acabamos de sugerir. Escolhemos o módulo Avaliação, uma das partes do Portal do Aluno, para expor seus dados e serviços seguindo parte da arquitetura REST conforme detalhamos acima. Esperamos que o artefato criado ajude a mapear dificuldades e vantagens da maneira como pretendemos implementar os passos que delineamos. Restringir este exemplo a apenas um módulo torna possível criar algo mais próximo, em termos de funcionalidades, de um módulo real do Portal do Aluno. Esperamos também com isso não extender demasiadamente o escopo deste trabalho.

A simplicidade do módulo Avaliação, se comparado com outros constituintes do Portal, faz com que ele seja um candidato interessante para usarmos como exemplo. Primeiro por não possuir regras de negócio complexas que tornariam a criação deste artefato mais demorada. Em segundo lugar por ter poucas dependências com relação aos outros componentes e serviços do Portal. Evitar estas regras de negócio mais complicadas neste momento não implica em perda de aplicabilidade do nosso trabalho aos outros constituintes do SIGA. Isto se deve ao fato de que estas funcionalidades seriam adionadas ao modelo de nossa aplicação sem precisar de alterarações significativas na forma como implementamos a camada acima delas, que é responsável por servir seus resultados. Uma vez que optamos por continuar a usar a linguagem *Java* não precisamos também traduzir a implementação existente para outra linguagem, o que ajuda a manter um baixo custo para este exemplo.

A forma usada para criar o artefato detalhada a seguir é aplicável a todos os sistemas mantidos pela equipe. Isto se deve ao fato de que a implementação da lógica das funcionalidades presentes na camada de modelo destes projetos não precisa ser alterada de fato. Adicionaremos uma nova camada sobre elas que servirá seus resultados, os convertendo para o formato JSON. Este ponto poderá ser melhor apreciado a seguir quando tratarmos dos aspectos mais técnicos do exemplo que criamos. Antes de abordar estes detalhes gostaríamos de ressaltar que o Portal do Aluno é um conjunto de aplicações de tamanho considerável. Ele utiliza várias ferramentas e foi desenvolvido ao longo de anos. Para que uma versão completa fosse desenvolvida seria necessário que algumas funcionalidades a mais fossem contempladas pelo nosso estudo. Esta replicação completa de todo o Portal do Aluno não é o nosso objetivo nessa demonstração. Alguns serviços do Portal fazem uso de restrições de acesso que são mediadas tanto pelo perfil que o usuário possui (aluno, professor e coordenador de curso são alguns exemplos) quanto pela localização na qual ele está alocado (Centro de Tecnologia na ilha do Fundão por exemplo), sendo possível também mediar o acesso de alguns serviços por meio de um calendário. Não cobrimos

esta capacidade aqui neste momento. Outra parte fundamental que não tocamos neste trabalho é a composição da interface gráfica utilizada pelos usuários, aqui nos preocupamos apenas com um conjunto restrito das funcionalidades do *backend* da aplicação.
A seguir delineamos alguns detalhes que são comuns a todos os módulos do Portal do Aluno e os comparamos com a implementação que criamos. Com este comparativo esperamos explicar melhor as diferenças entre o ponto de onde partimos, composto pela organização atual do Portal do Aluno, para o resultado final que obtivemos neste trabalho.

5.1 PRINCIPAIS ARCABOUÇOS DO PORTAL DO ALUNO

Tratando-se das ferramentas usadas no Portal do Aluno, um de seus arcabouços centrais é o chamado *Seam Framework*[1]. Dentre suas responsabilidades está a gerência do estado da aplicação, ou seja, controlar o ciclo de vida dos objetos *Java* que a compõe desde sua criação até o momento em que são destruídos. Para que isto seja feito é preciso declarar tais objetos como um componente *Seam*, onde a forma mais utilizada nestes projetos que abordamos é demarcando a classe que o define com a anotação *@Name*. Este arcabouço também implementa o conceito de escopo para os objetos, determinando dessa forma até quando um componente gerenciado por ele deverá existir. Estes escopos são controlados pela anotação *@Scope*. Alguns exemplos de escolhas para a duração dos escopos são: evento, página, sessão e aplicação. Assim o desenvolvedor pode escolher perdurar os objetos instanciados somente até o fim de uma requisição para uma página ou serem mantidos durante todo o tempo em que a aplicação estiver em execução, podendo escolher dentre outras granularidades que se enquadram entre essas duas. Um uso comum deste mecanismo ocorre nos formulários[2] preenchidos e submetidos por usuários, como por exemplo na escolha das ofertas de disciplinas durante o período de inscrição. Alguns formulários podem ocupar mais de uma página ou necessitar de múltiplas requisições, sendo possível manter nos servidores da aplicação o estado destes objetos com os valores que foram repassados ao longo da interação do usuário com o serviço.
Outra capacidade que o arcabouço *Seam* provê é a injeção de dependências, amplamente utilizada nos projetos do Portal. Com este mecanismo ao invés de criarmos um objeto novo usando a palavra reservada *new* como de costume na linguagem *Java* podemos utilizar a anotação *@In*. Assim o arcabouço irá buscar por um objeto deste tipo que já se encontra gerenciado por ele e atribuir esta instância ao campo em questão em que usamos tal anotação. Um exemplo comum de onde as capacidades citadas acima são empregadas ocorre nos filtros do Portal do Aluno, ilustrado na figura 5. Após realizar uma consulta em um filtro como este o usuário pode então selecionar logo abaixo um dos resultados obitidos, note que caso ele escolha retornar ao filtro os resultados da consulta realizada

[1] <https://www.seamframework.org/>
[2] <https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/form>

anteriormente ainda estarão presentes. Isto se deve ao mecanismo de escopo que citamos, ele determina que a aplicação mantenha o estado obtido na consulta inicial sem que haja necessidade de buscar estas informações novamente no banco de dados. Mais detalhes sobre os componentes do *Seam* e sobre a injeção de dependências em geral podem ser encontrados na *web*[3,4].

Como os mecanismos acima são parte integral de como os serviços do Portal do Aluno são implementados é desejável obtermos substitutos para eles. Estes devem oferecer estas capacidades ou ao menos outras semelhantes o suficiente a fim de amenizar a tarefa de reimplementar as funcionalidades existentes. Outro arcabouço amplamente utilizado no Portal é o *Hibernate ORM*, responsável pelo mapeamento dos dados na forma de objetos *Java* para o paradigma relacional presente no banco de dados. Esta diferença é endereçada utilizando o chamado mapeamento objeto-relacional (do inglês *Object-Relational Mapping* ou ORM). Nele podemos definir, através de anotações da linguagem Java ou arquivos XML, as características dos relacionamentos entre as entidades como por exemplo sua cardinalidade: um para um, um para muitos, muitos para muitos. Podemos demarcar também quais colunas são usadas nas ligações entre tabelas. Outras funcionalidades ofertadas por este arcabouço são a validação dos dados recebidos e o gerenciamento de transações que encapsulam as chamadas dos métodos presentes nos DAO. Mais detalhes sobre o arcabouço podem ser encontrados em sua página na *web*[5], para um outro exemplo de uso do *Hibernate ORM* consulte (O'NEIL, 2008). Decidimos manter o *Java* como a linguagem escolhida para o projeto pois caso contrário precisaríamos reconsiderar todo o conjunto de ferramentas e arcabouços utilizados, o que tornaria o trabalho como um todo mais custoso. As principais mudanças levando em consideração as ferramentas foram a atualização da versão da linguagem e a substituição do arcabouço *Seam*. Com o intuito de reaproveitar ao máximo o trabalho já executado na criação do Portal do Aluno mantivemos o uso dos mapeamentos objeto-relacional e o arcabouço *Hibernate ORM*, uma vez que ele atende satisfatoriamente as necessidades atuais da aplicação.

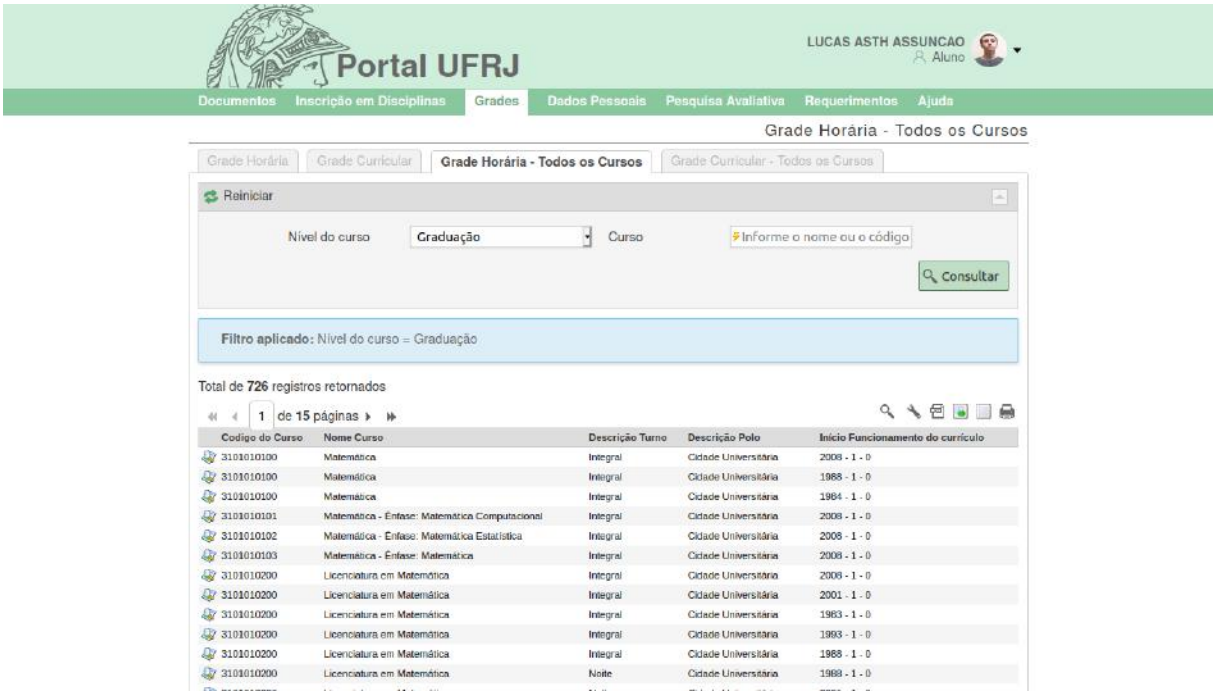
A substituição do *Seam* foi considerada por dois motivos: primeiro por ele não estar em desenvolvimento ativo e segundo por haver a possiblidade de substituí-lo por outro arcabouço com um conjunto de capacidades similares. Detalharemos em seguida o substituto escolhido para o arcabouço *Seam*.

5.2 SUBSTITUINDO O ARCABOUÇO *SEAM* PELO *SPRING*

Com o intuito de fornecer compatibilidade com a forma que a equipe desenvolve suas aplicações atualmente e abrir espaço para as mudanças que detalhamos acima propomos o arcabouço *Spring* como um substituto para o *Seam*. Devido ao seu amplo uso no Portal

[3] <https://docs.jboss.org/seam/2.3.0.Final/reference/en-US/html/concepts.html>
[4] <https://martinfowler.com/books/eaa.html>
[5] <https://hibernate.org/orm/>

Figura 5 – Exemplo de um filtro do Portal do Aluno



Fonte: figura do autor

tal substituição demandaria um esforço considerável. Todavia se levarmos em conta a interseção entre as capacidades destes arcabouços podemos manter boa parte da estrutura criada no Portal, substituindo os mecanismos usados atualmente pelos seus correspondentes no *Spring*. Uma das funcionalidades mais utilizadas do arcabouço *Seam* no Portal do Aluno é a injeção de dependências, conforme citamos anteriormente. No *Spring* também dispomos desta capacidade sendo invocada através da anotação *@Autowired*. No anexo F.1 temos dois exemplos de objetos que são instanciados através desta forma.

No arcabouço *Seam* demarcamos objetos que serão gerenciados por ele através da anotação *@Name*. O *Spring* também possui uma anotação com propósito similar chamada *@Component*. No anexo G.1 temos um exemplo da definição de um componente na implementação que criamos usando esta anotação. Maiores detalhes destes mecanismos podem ser encontrados na documentação do arcabouço[6]. Outro elemento utilizado frequentemente no Portal conforme citamos anteriormente é a anotação *@Scope* que define até quando uma determinada instância de um objeto estará disponível. Esta capacidade também está presente no *Spring* através de uma anotação de mesmo nome, porém em nossa implementação não fizemos uso desta capacidade. Isto se deve ao fato de seguirmos a arquitetura REST, o que implica na redução do estado mantido pela aplicação, que é resultado das requisições feitas aos servidores. Em nosso exemplo não foi necessário demarcar a duração do ciclo de vida destes objetos uma vez que eles não possuem es-

[6] <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework>

tado a ser propagado além de uma única requisição. Mesmo assim ter esta capacidade à disposição é um ponto positivo, uma vez que é possível encontrar um cenário futuro que demande seu uso.

Devido à presença das semelhanças acima no conjunto de capacidades oferecidas pelos dois arcabouços acreditamos que o *Spring* seja uma boa escolha para replicarmos o funcionamento das aplicações que compõe o Portal do Aluno. Além destas similaridades os diversos projetos que compõe este arcabouço possuem outras funcionalidades que utilizamos para criar nossa implementação e facilitar o trabalho do desenvolvedor. Destes o principal projeto que usamos é o *Spring Data JPA*[7], através dele podemos definir as consultas SQL mais simples de maneira declarativa. Assim suprindo a necessidade de definí-las nos arquivos XML como feito no Portal. O mecanismo usado para alcançar isto é o chamado *repository*, um exemplo dele voltado para a classe que representa a entidade avaliação em nosso exemplo pode ser encontrado no anexo E.1. Note que neste anexo também fazemos uso da anotação *@Query*, nela declaramos as consultas mais extensas que não são contempladas pela maneira declarativa que citamos. Um exemplo que ilustra estas consultas mais simples é representada pelo método *findAvaliacaoByOid* neste mesmo anexo. Note que não precisamos especificar a consulta, apenas pela assinatura do método o arcabouço é capaz de gerar a consulta equivalente. Maiores detalhes desta capacidade podem ser encontrados na documentação do arcabouço[8].

5.3 CRIANDO A BASE DO PROJETO

O primeiro passo na criação do nosso exemplo foi gerar a base do projeto. Isto foi feito utilizando um gerador de projetos disponível na seguinte página da *web*[9]. Nela é possível customizar esta base podendo escolher diversas dependências do projeto que podem ser constituintes do próprio arcabouço *Spring* ou não. Dentre os arcabouços que compõe o projeto *Spring* fazemos uso do *Spring Data JPA*, *Spring Boot* e *Spring Security*. Sendo o primeiro usado na interação com a camada de persistência de dados, o segundo para prover a estrutura e configurações básicas do projeto e o último para autenticação e autorização dos usuários na aplicação. Detalhes destes projetos e suas documentações podem ser consultadas em suas respectivas páginas[10].

A gerência de dependências do projeto é feita utilizando uma ferramenta chamada *Maven*. Aqui devemos apontar uma diferença substancial entre a nossa implementação e a versão original do projeto. Na versão atual do Portal as dependências são gerenciadas pela própria equipe, usando um repositório local. Na nossa implementação estamos buscando

[7] <https://spring.io/projects/spring-data-jpa>

[8] <https://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html>

[9] <https://start.spring.io/>

[10] <https://spring.io/projects>

estas de um repositório que é mantido por terceiros[11]. Ambas as escolhas de ferramentas, nossa e do SIGA, contam com suporte do ambiente integrado de desenvolvimento (do inglês *Integrated Development Environment* ou IDE) de escolha da equipe, que no caso é o *Eclipse IDE* [12]. Utilizamos a mesma IDE durante o desenvolvimento da nossa versão.

5.4 MÓDULOS ADICIONAIS NECESSÁRIOS

Durante o desenvolvimento do Portal do Aluno além do módulo em que se deseja alterar são necessários outros módulos adicionais. Em nosso caso além do projeto que contém o módulo Avaliação, para que ele funcione adequadamente precisamos de outros dois adicionais. Os projetos *Commons* e Segurança detém funcionalidades que são necessidades comuns a todos os outros módulos do Portal. Exemplos destas são o mapeamento ORM das entidades da aplicação e os mecanismos de autorização e autenticação de usuários, providos pelo *Commons* e pelo Segurança respectivamente. Para implementarmos nosso exemplo precisamos de algumas funcionalidades básicas servidas por estes dois projetos e portanto precisamos recriá-los também. Ao criar nossa própria versão destes módulos adicionais optamos por fazer algumas mudanças.

Referente ao *Commons* podemos listar duas mudanças significativas que fizemos com relação ao original. Em primeiro lugar removemos os arquivos XML usados pelo *Hibernate* para o mapeamento de entidades e armazenamento de consultas SQL. Utilizamos apenas a forma declarativa de se fazer tais mapeamentos que é através de anotações da linguagem *Java*. A segunda mudança que gostaríamos de ressaltar no módulo *Commons* é referente às classes que definem as entidades do Portal. Nelas utilizamos uma ferramenta para simplificar a sua definição, reduzindo o seu tamanho e melhorando sua legibilidade. Para acessarmos os campos dos objetos definidos por estas classes usamos métodos específicos chamados *getters* e *setters*. Na programação orientada a objetos o motivo para a existência de tais métodos é abstrair a forma como tais valores são buscados ou alterados. O que ocorre em nosso caso é que muitas vezes estes métodos simplemente acessam e alteram os valores dos campos sem realizar nada além da própria atribuição ou busca dos valores. Para evitar estes casos e reduzir o tamanho das classes usamos o *Lombok* [13]. Ele é uma biblioteca *Java* voltada para automação de certas tarefas do desenvolvedor, dentre elas a criação dos *getters* e *setters* que citamos. Usando as anotações *@getter* e *@setter* nos campos para os quais desejamos que sejam criados estes métodos. As diferenças entre a versão nova utilizando o *Lombok* e a original podem ser observadas comparando os anexos A.1 e B.1 que contém estas respectivas versões da entidade avaliação. Para os raros casos onde é necessário se definir alguma lógica sobre o acesso dos campos das entidades podemos voltar a definí-los da forma como era feito antes, especificando tais métodos

[11] <https://repo1.maven.org/maven2/>
[12] <https://www.eclipse.org/eclipseide/>
[13] <https://projectlombok.org/>

manualmente como exemplificado no anexo B.1.

Em nossa implementação do módulo Segurança fizemos mudanças mais drásticas. Reforçamos o fato que ela não contém todas as funcionalidades das quais a versão em uso no Portal do Aluno atualmente dispõe. Por termos escolhido substituir o *Seam* foi necessário alterar a forma como é feita a autenticação do usuário e a autorização de suas ações. Isto se deve ao fato de a implementação destes mecanismos no Portal se basearem em integrações deste arcabouço com outras ferramentas que não consideramos neste trabalho. Outro motivo para criarmos esta nova implementação foi tornar os testes que realizamos mais próximos da realidade. Uma vez que os mecanismos de segurança impõem um incremento na latência das requisições. Por optarmos pelo arcabouço *Spring* precisamos abandonar a forma anterior de tratar a segurança da aplicação. Devido ao conjunto de ferramentas que escolhemos a opção mais simples foi utilizar o *Spring Security*. Ele é a parte do conjunto de arcabouços que compõe o projeto *Spring* voltada para os requisitos de segurança das aplicações. No nosso caso podemos destacar duas ocasiões onde usamos esta ferramenta. Primeiro na criação de uma nova versão do módulo segurança, este projeto é uma aplicação separada contendo um *endpoint* onde o usuário que deseja interagir com nossa implementação deve primeiro realizar o processo de obtenção de um *token* de acesso. Em nosso caso optamos por usar o padrão *JSON Web Token* (JWT) para gerar este *token*. Nele estão agrupados dados como o identificador do usuário ao qual ele se aplica e quais as permissões que existem para este usuário no momento em que o *token* foi emitido. A segunda ocasião onde fazemos uso uso do *Spring Security* é no esquema de autorização básica que criamos, que é baseado nos perfis que foram concebidos ao usuário. Este mecanismo de verificação dos perfis fica na definição dos *endpoints* da aplicação, conforme exemplificado no anexo F.1. Usando a anotação *@PreAuthorize(hasPermission('manutenção'))* estamos permitindo que apenas usuários com o perfil de manutenção possam interagir com o ponto de acesso onde adicionamos esta anotação. Note que esta permissão não condiz com o perfil de usuário que normalmente acessaria os *endpoints* relacionados ao anexo que citamos. Estas permissões foram definidas com caráter temporário para testarmos o mecanismo, podendo ser alteradas futuramente conforme necessário.

Para o versionamento do código fonte dos projetos que criamos foi utilizado o *Git* [14], sendo esta a mesma ferramenta utilizada no caso do Portal. Todos os artefatos que criamos neste trabalho podem ser consultados acessando o seu repositório na *web*[15].

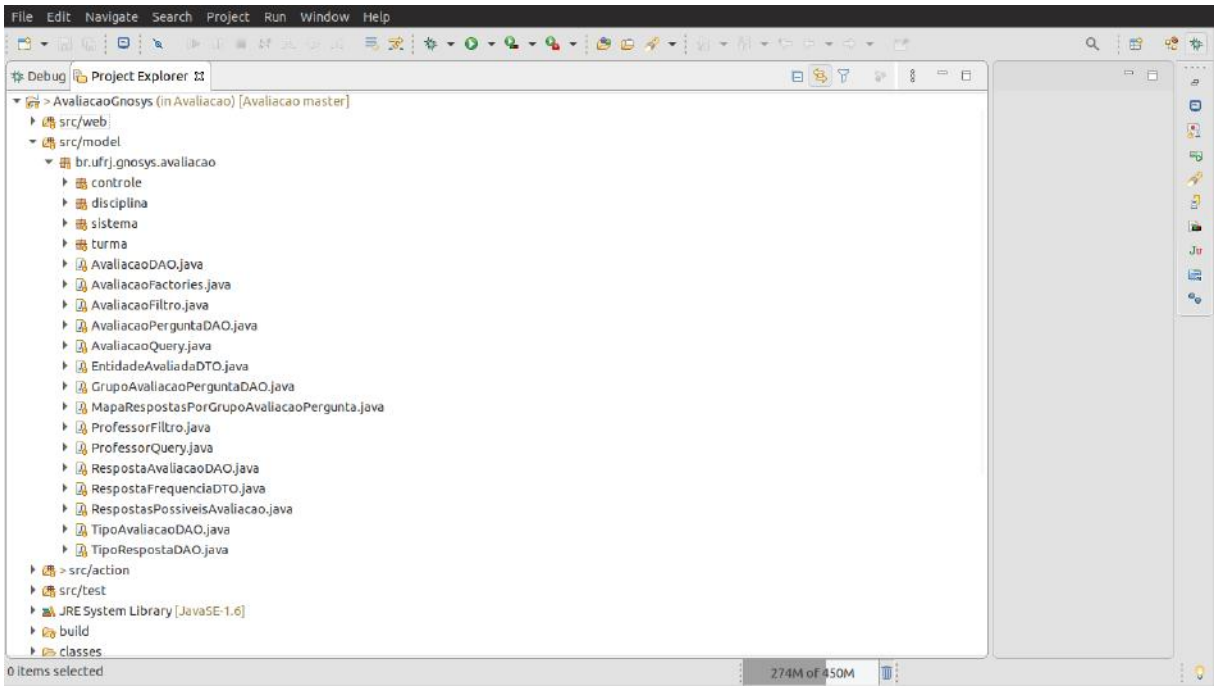
5.5 IDENTIFICANDO E EXPONDO SERVIÇOS EM NOSSO EXEMPLO

Detalhamos no começo deste capítulo formas de identificar os serviços e suas entidades envolvidas em cada módulo do Portal do Aluno. Podemos exemplificar estes passos usando o módulo que escolhemos, no caso o Avaliação, para criar um exemplo de implementação.

[14] <https://git-scm.com/>
[15] <https://gitlab.com/lcsa/tcc>

Na figura 6 é possível observar a estrutura do módulo Avaliação, mais especificamente sua camada de modelo.

Figura 6 – Versão original do módulo Avaliação no ambiente integrado de desenvolvimento



Fonte: figura do autor

Determinanos quais são as classes contendo os DAO associados a este projeto utilizando a busca por nome de arquivos fornecida pela IDE que usamos, no caso o *Eclipse IDE*. Desta forma descobrimos que temos, dentre outros, os seguintes DAO: AvaliacaoDAO, AvaliacaoPerguntaDAO, GrupoAvaliacaoPerguntaDAO, RespostaAvaliacaoDAO, TipoAvaliacaoDAO e TipoRespostaDAO. Assim já podemos identificar quais entidades estão envolvidas neste projeto, buscando pelas definições de suas respectivas classes conseguimos ter uma idéia melhor do propósito de cada uma delas. Neste caso temos a entidade Avaliacao que representa uma avaliação que será respondida por um usuário do sistema. Na definição dela podemos perceber que ela possui um conjunto de grupos de perguntas, representados por uma lista de objetos criados a partir da classe GrupoAvaliacaoPergunta. Cada grupo possui seu conjunto de perguntas que serão respondidas pelos avaliadores, estas são representadas pela lista de entidades do tipo AvaliacaoPergunta. Estas perguntas por sua vez possuem um objeto do tipo TipoResposta que indica o formato que terá a resposta desta pergunta. Exemplos deste tipo de resposta podem ser um campo de texto livre ou uma resposta baseada em concordância, com seletores expondo opções que vão de discordo totalmente até concordo totalmente passando por diversas opções entre estas. Para cada tipo de avaliação temos uma entidade chamada TipoAvaliacao, ela corresponde ao tipo com o qual ela foi cadastrada. Os tipos de avaliação

são: do sistema, de turma e de disciplina. Eles possuem as entidades RespostaAvaliacaoSistema, RespostaAvaliacaoTurma e RespostaAvaliacaoDisciplina associadas a cada um deles respectivamente. Na tabela 3 temos alguns exemplos dos pontos de acesso da API que criamos onde estas entidades podem ser consultadas utilizando o identificador da avaliação da qual eles fazem parte. Na figura 7 temos um exemplo de tela exibindo uma avaliação no Portal do Aluno.

Tabela 3 – Pontos de acesso para algumas entidades

$/Avaliacao/avaliacao/\{avaliacao\_oid\}$
$/Avaliacao/avaliacaoPergunta/avaliacao/\{avaliacao\_oid\}$
$/Avaliacao/grupoAvaliacaoPergunta/avaliacao/\{avaliacao\_oid\}$
$/Avaliacao/respostaAvaliacao/respostaDisciplina/\{avaliacao\_oid\}$
$/Avaliacao/respostaAvaliacao/respostaTurma/\{avaliacao\_oid\}$
$/Avaliacao/respostaAvaliacao/respostaSistema/\{avaliacao\_oid\}$

Fonte: tabela do autor

Para o módulo do Portal que escolhemos neste exemplo os serviços identificados não agregam muitas operações sobre as entidades que listamos. Se olharmos para o fluxo de criação, edição e resposta das avaliações em nosso exemplo podemos perceber que todas estas etapas podem ser feitas interagindo diretamentamente com os *endpoints* específicos para cada uma das entidades identificadas acima. Sendo assim no caso do módulo avaliação, para as funcionalidades atualmente em uso dele, observar as classes referentes aos DAO contidos nele e suas entidades associadas é o suficiente. Para cada uma delas criamos uma interface que pode ser acessada tanto pelos navegadores quanto por outros agentes. Desta forma a camada de apresentação da aplicação fica responsável por organizar os dados obtidos através dos *endpoints* destas interfaces que fornecemos, para que ao final o usuário possa utilizá-los de maneira satisfatória.

5.6 ESTRUTURA GERAL DE NOSSO EXEMPLO DE IMPLEMENTAÇÃO

Podemos resumir a estrutura do exemplo que criamos detalhando 3 partes distintas de nossa implementação. Os controladores REST, seguidos das classes responsáveis pela conversão e validação dos dados e finalmente os repositórios do *Spring*. Na figura 8 temos um diagrama contendo o fluxo de uma requisição relacionada aos componentes responsáveis por avaliações. Os controladores contém métodos específicos (chamados *handlers*) para as ações que podem ser realizadas com uma avaliação. Dependendo de qual método (também chamado *handler* nesse contexto) a requisição ativar pode ser necessário validar os dados, como no caso do cadastro de uma avaliação. Para cenários mais simples como

buscar uma avaliação específica o controlador pode interagir com o repositório diretamente. Neste exemplo a classe *CadastroAvaliacaoBean* exemplifica um componente que usamos para converter os dados do formato JSON para uma classe *Java*. Apesar de não ser este o caso podemos incluir junto a este componente regras de negócio e ou de validação dos dados vindos do usuário. Após convertidos e validados estes dados, agora na forma de uma classe, são repassados para o seu respectivo repositório, que são classes que utilizam os mecanismos do arcabouço *Spring* para lidar com as atribuições de persistência da aplicação.

Fazendo um paralelo com o padrão de projeto MVC podemos observar onde cada um destes componentes se encaixa de acordo com este padrão. No exemplo dado o *AvaliacaoController* é caracterizado como um controlador, detendo métodos relacionados a ações que os usuários podem requisitar. Classes como o *CadastroAvaliacaoBean* e *AvaliacaoRepository* pertencem ao modelo, por deter as funcionalidades que lidam com o acesso ao banco de dados e transformação dos dados da aplicação.

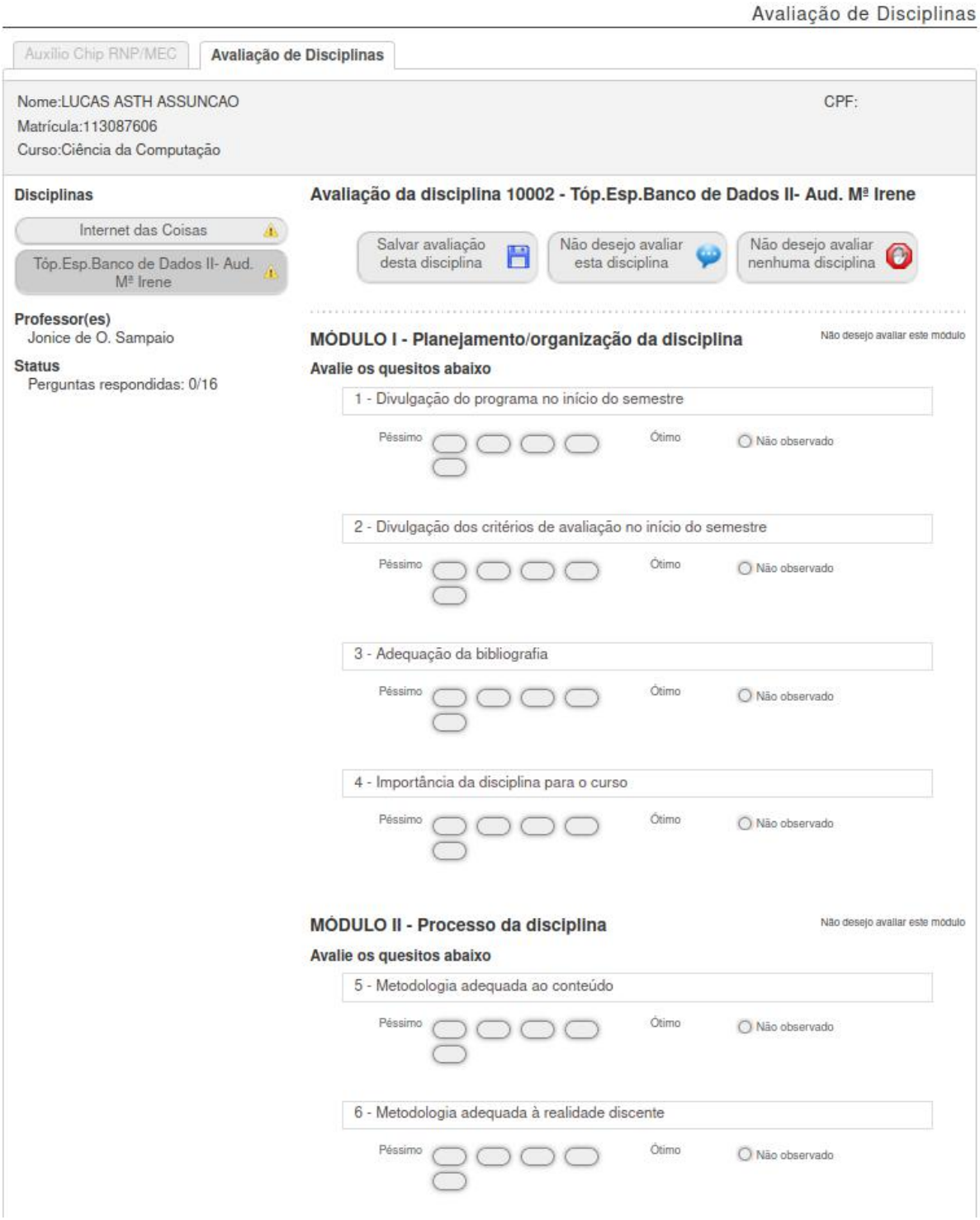
5.7 EXEMPLO DE USO DE NOSSA IMPLEMENTAÇÃO

Tendo apresentado e comparado as ferramentas usadas tanto no Portal quanto por nós gostaríamos de ilustrar o uso dos artefatos que criamos. Primeiro precisamos criar uma instância de um banco de dados para nossa aplicação. Isto pode ser feito de diversas formas, em nosso caso usamos um *container Docker*[16] contendo um banco de dados *MySQL*[17]. É necessário criar diretamente neste banco o usuário que será usado para interagir com a aplicação. Antes de criar o usuário é necessário popular as tabelas usadas em sua definição, sendo estas *Pessoa* e *PessoaFisica*. Após criar o usuário precisamos atribuir uma permissão para que ele consiga interagir com a aplicação. Em nosso caso atribuímos a ele uma permissão com o perfil de usuário de manutenção, para que assim ele esteja autorizado a interagir com todos os pontos de acesso da API que criamos. O primeiro passo para interagir com ela é obter um *token* de acesso, que nosso caso é um *JSON Web Token*. Para obtê-lo é necessário realizar um POST HTTP contendo as credenciais do usuário (mantidas no corpo da requisição) para o *endpoint* com caminho */Seguranca/auth*. Caso as credenciais sejam aceitas a resposta da requisição irá conter o *token* JWT.
Em nosso exemplo vamos supor que um usuário queira consultar o *endpoint* responsável por avaliações, a fim de obter uma avaliação em especial. Todos os pontos de acesso que criamos da *API* (com exceção daquela relacionado ao *logout*) necessitam que o usuário esteja autenticado. Portanto o primeiro passo para o usuário deve ser se autenticar realizando um *POST* em */Seguranca/auth* no módulo Segurança informando no corpo da requisição suas credenciais que neste caso são o identificador do usuário e sua senha. Caso

[16] <https://www.docker.com/>
[17] <https://www.mysql.com/>

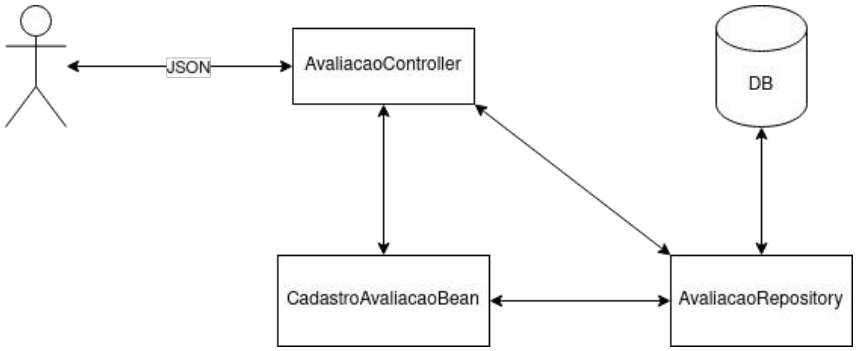
as informações repassadas estejam corretas a *API* irá retornar um token *JWT*. Este *token* deve ser enviado junto a todas as requisições subsequentes, sendo passado no *header* HTTP denominado *Authorization*. Assim o usuário poderá então consultar o *endpoint* desejado conforme a sequência exemplificada na figura 9. Ao obter uma avaliação o usuário terá acesso a suas perguntas, podendo em seguida cadastrar suas respostas referentes à avaliação que ele buscou.

Figura 7 – Tela para preenchimento de avaliação



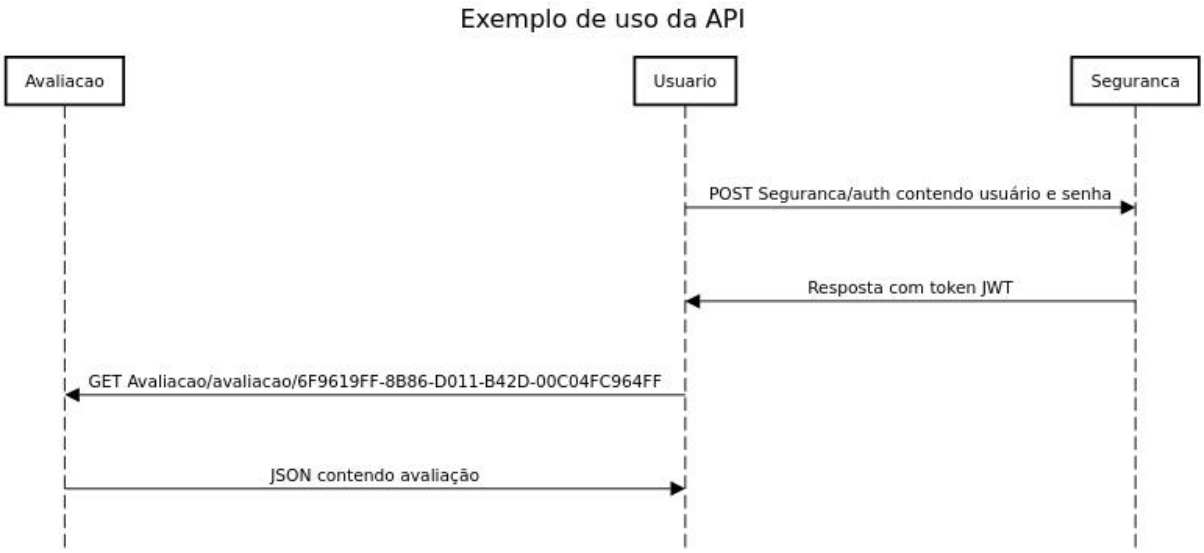
Fonte: figura do autor

Figura 8 – Principais elementos da implementação



Fonte: figura do autor

Figura 9 – Obtenção do *token* e acesso a um *endpoint*



Fonte: figura do autor

6 RESULTADOS

Nesta seção apresentamos efeitos, tanto os desejados quanto negativos, causados por nossa implementação. Estes merecem ser considerados ao final desta primeira iteração e também durante uma possível continuação deste trabalho. Ressaltamos alguns aspectos técnicos das soluções para problemas que surgiram ao desenvolver nossa implementação. Expomos também resultados referentes ao comportamento da aplicação com relação ao uso de recursos.

6.1 AVALIAÇÃO DO DESEMPENHO

Com o intuito de obtermos uma primeira impressão do consumo de recursos do exemplo que criamos foi realizado um teste de carga. Usamos uma ferramenta chamada *Locust*[1] para gerar as requisições necessárias com destino à nossa aplicação. Os arquivos referentes ao carregamento do cenário e roteiro de execução do teste podem ser consultados no repositório deste trabalho na *web*[2]. O cenário que construímos para este teste simula as requisições feitas por alunos ao obter um *token* para se autenticar na aplicação, buscar por uma avaliação de disciplinas específica e em seguida cadastrar suas respostas. O tempo de execução deste teste foi de cerca de 10 minutos. Foram simulados no total 1000 usuários onde cada um deles leva entre 20 a 60 segundos para cadastrar suas respostas.
Dos recursos computacionais que utilizamos para execução do teste tanto os processos da ferramenta geradora de carga quanto as aplicações alvo e seu banco de dados compartilharam a mesma máquina hospedeira. Esta configuração não corresponde ao cenário real onde estes elementos estariam em execução. Mesmo assim acreditamos que ela seja suficiente para obtermos uma primeira impressão e indicar a direção geral do comportamento dos nossos artefatos.
Um ponto importante a se ressaltar referente ao agrupamento de todos os elementos do teste em uma única máquina é a latência experimentada pela requisições devido ao tráfego na rede. Como esta latência não foi experimentada neste caso, por não haver necessidade de tráfego entre máquinas, haverá uma diferença no comportamento da aplicação quando for utilizado o ambiente onde esta separação ocorre. Nossa expectativa é que nele as conexões criadas permaneçam abertas durante um período de tempo maior. Isto seria devido a latência das requisições que estaria sujeita à condições da rede no momento. Isto causaria um uso de memória maior, devido ao maior número de conexões mantidas em aberto com nossas aplicações neste caso. Em contrapartida uma carga menor na unidade central de processamento (CPU) do hospedeiro seria experimentada, devido aos maiores intervalos

[1] <https://locust.io/>
[2] <https://gitlab.com/lcsa/tcc/>

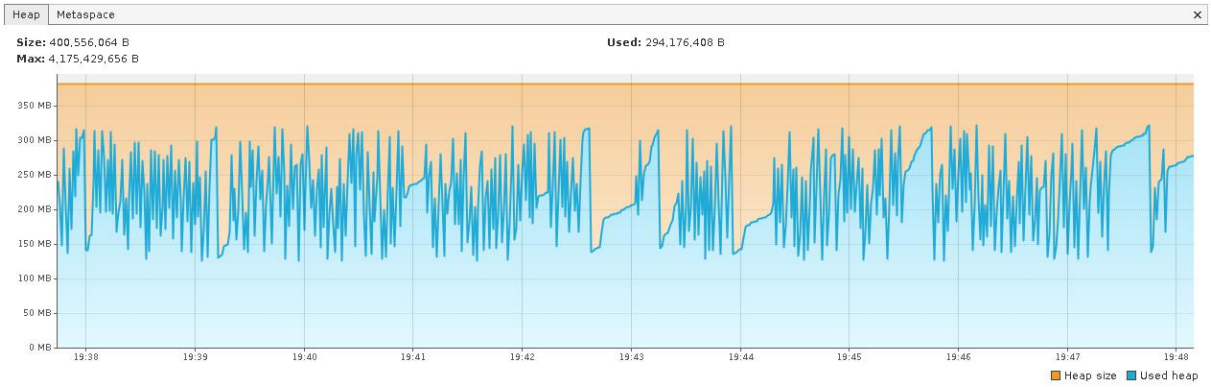
entre chegadas de requisições ao servidor. Podemos constatar pelos resultados obtidos que o consumo de memória de nosso exemplo possui caráter promissor, conforme mostrado nas figuras 10 e 11. Estas métricas foram acompanhadas usando uma ferramenta chamada *VisualVM*[3], ela provê uma interface gráfica que apresenta algumas métricas expostas pela JVM. Na tabela 4 temos a especificação da máquina que utilizamos para executar o teste de carga.

Tabela 4 – Recursos utilizados no teste de carga

Tipo de Recurso	Instância
Processador	Intel(R) Core(TM) i5-9600K CPU @ 3.70GHz
Memória	16 GB 2400 MHz
Armazenamento	SSD WD Green M.2 2280 240GB
Sistema Operacional	GNU/Linux 5.15.12-100

Fonte: tabela do autor

Figura 10 – Heap alocada durante o teste para nossa versão do módulo Avaliação

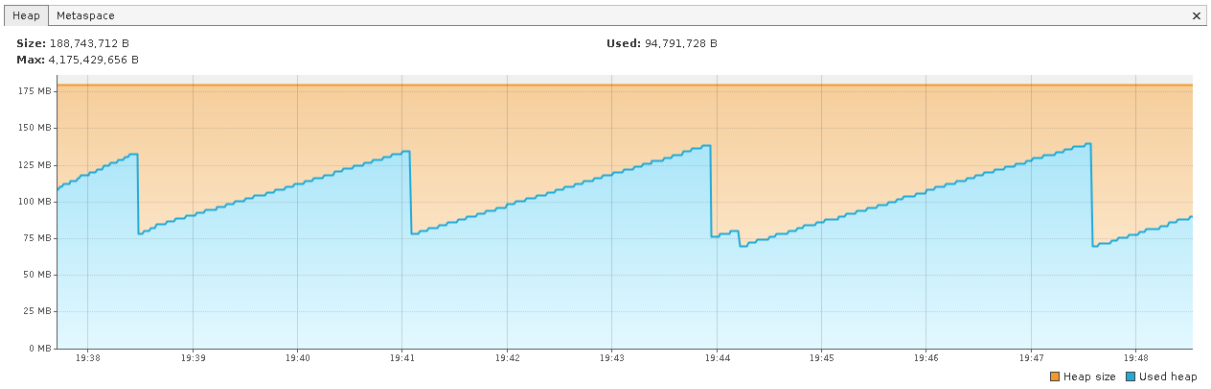


Fonte: figura do autor

Concluímos que o consumo de memória é adequado, uma vez que em nossa implementação do módulo Avaliação consumimos 400 MB e na versão atual no Portal do Aluno são usados cerca de 1,8 GB. Apesar de ser importante esta não é a única métrica relevante. No que tange ao tempo de resposta das requisições feitas observamos, conforme ilustrado nas figuras 13 e 14, uma quantia de tempo relativamente alta. No caso do cadastro de respostas levando em média 3,6 segundos para cadastrar uma resposta. Na figura 15 podemos perceber que o consumo de CPU se manteve alto durante a execução do teste, chegando em alguns momentos ao limite disponível na máquina que hospedou os testes. O uso de CPU durante o teste pode ser acompanhado pela figura 12. Isto certamente teve um impacto negativo no tempo de resposta das requisições, devido a aplicação estar competindo por CPU com o banco de dados local e também com os processos da ferramenta responsável por gerar a carga de nosso teste. Apesar disso a taxa de erros se

[3] <https://visualvm.github.io/>

Figura 11 – Heap alocada durante o teste para nossa versão do módulo Segurança

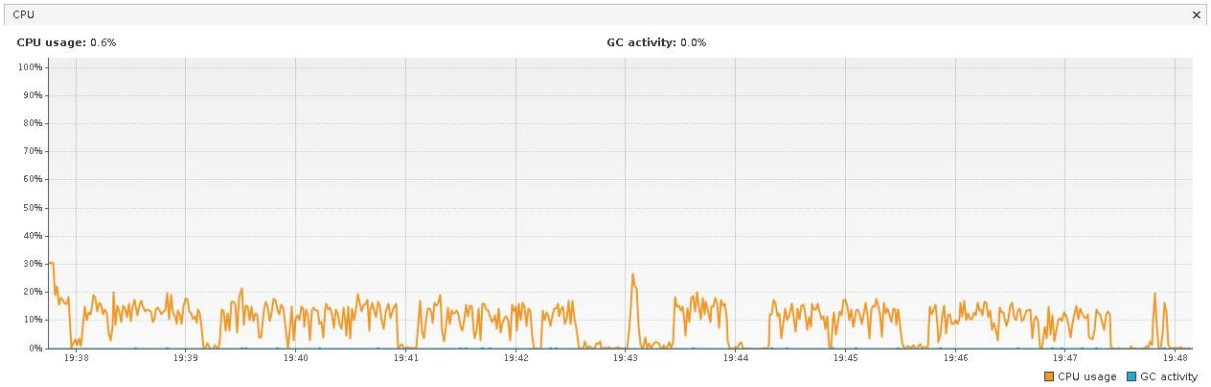


Fonte: figura do autor

manteve baixa, com um total de 11 falhas para cadastro de respostas em um total de 189376 requisições.

Mesmo que os resultados aparentem ser suficientes, se comparado ao comportamento da versão atual da aplicação, é desejável que este roteiro de teste seja executado novamente no futuro. Onde, dispondo de mais recursos, possamos separar em máquinas diferentes a aplicação, o banco de dados e os processos do *Locust*. De forma a obter um resultado mais próximo do esperado para o ambiente de produção.

Figura 12 – Uso de CPU durante o teste para nossa versão do módulo Avaliação



Fonte: figura do autor

Figura 13 – Resumo de resultados do teste de carga

Locust Test Report

During: 2022-02-19 19:36:15 - 2022-02-19 19:47:49

Target Host: http://localhost

Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
POST	/Avaliacao/respostaAvaliacao /respostaDisciplina	189376	11	3593	37	34363	46	273.1	0.0
POST	/Seguranca/auth	1001	1	95	3	943	0	1.4	0.0
	Aggregated	**190377**	**12**	**3575**	**3**	**34363**	**46**	**274.5**	**0.0**

Response Time Statistics

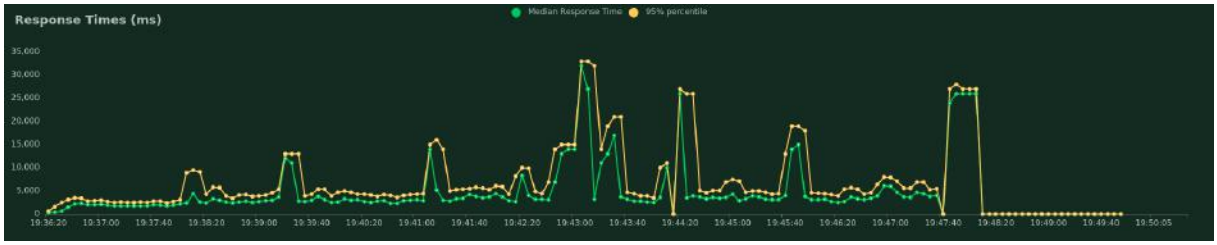
Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
POST	/Avaliacao/respostaAvaliacao /respostaDisciplina	2700	3000	3400	4000	5000	8100	26000	34000
POST	/Seguranca/auth	37	57	87	140	250	400	690	940
	Aggregated	**2700**	**3000**	**3400**	**4000**	**5000**	**8000**	**26000**	**34000**

Failures Statistics

Method	Name	Error	Occurrences
POST	/Seguranca/auth	HTTPError('403 Client Error: for url: http://localhost:8081/Seguranca /auth')	1
POST	/Avaliacao/respostaAvaliacao/respostaDisciplina	HTTPError('500 Server Error: for url: http://localhost:8082/Avaliacao /respostaAvaliacao/respostaDisciplina')	11

Fonte: figura do autor

Figura 14 – Tempo de resposta das requisições do teste



Fonte: figura do autor

Figura 15 – Consumo de CPU da máquina hospedeira durante o teste

Fonte: figura do autor

6.2 PROBLEMAS ENCONTRADOS EM NOSSA SOLUÇÃO

6.2.1 Referências circulares no JSON gerado pela API

Nas definições das entidades da aplicação podemos identificar relacionamentos onde ocorrem referências cíclicas. No módulo Avaliação podemos usar como exemplo a entidade avaliação, ela contém uma lista dos grupos de perguntas contidos nesta avaliação. Ao olhar a definição da classe desses objetos (*GrupoAvaliacaoPergunta.java*) podemos perceber que cada um destes grupos contém uma referência para a classe que representa uma avaliação. Referências deste tipo representam um problema em nossa implementação, pois ao transformar estes objetos para o formato JSON precisamos definir um ponto de parada ao seguir estas referências. Caso isto não seja feito a biblioteca que utilizamos seguirá elas, aninhando os objetos até que ocorra uma exceção na aplicação.
Precisamos evitar que as estruturas sejam repetidas até que o limite da pilha da Máquina Virtual Java (JVM) seja atingido, causando uma exceção de estouro de pilha (*StackOverflowError*[4]). Para tal demarcamos os campos que compõe estas referências circulares com uma anotação específica. Utilizando a anotação *@JsonIdentityInfo*[5] definimos que ao ser encontrada pela segunda vez, a referência ao objeto não deve ser serializada por completo, populando o campo referente ao objeto apenas com o seu identificador único (chamado OID ou *Object Identifier*) e parando de seguir a referência neste ponto. Dessa forma impedimos que o processo de serialização destes objetos entre em recursão indefinidamente.

6.2.2 Incompatibilidade com a camada de visualização atual

Ao abordar as tecnologias utilizadas na versão atual do Portal do Aluno citamos o arcabouço *Seam*. Integrado a ele está o *Java Server Faces* (JSF), o principal elemento usado na composição de interfaces gráficas para o Portal do Aluno. Ao substituir o arcabouço *Seam* torna-se ncecessário encontrar um substituto para o JSF, uma vez que este está integrado a ele. Neste cenário poderíamos aproveitar alguns elementos como folhas de estilo (CSS) e arquivos contendo as definições das páginas em HTML. Mesmo com estes pontos uma transição do cenário atual partindo para um novo arcabouço que endereça as atribuições do *front-end* representaria uma tarefa extensa. Desta forma esta mudança onde abandonamos o *Seam* é custosa, se considerarmos este trabalho necessário para recriar as telas do Portal do Aluno.
Um outro ponto importante sobre a implementação que criamos é que ela coexistiria com a versão atual do Portal do Aluno. Seu custo em recursos, mesmo que de aparência afável, representa um consumo adicional ao que já ocorre atualmente. Desta forma não

[4] <https://docs.oracle.com/javase/10/docs/api/java/lang/StackOverflowError.html>
[5] <https://github.com/FasterXML/jackson-annotations/wiki/Jackson-Annotations>

poderíamos citar que ela trará uma diminuição deste consumo, pelo menos não enquanto as duas versões existirem lado a lado.

6.3 VISÃO GERAL DOS RESULTADOS

Mesmo tendo encontrado os problemas que ressaltamos acima acreditamos que nossa solução seja um bom ponto de partida. No sentido em que apresentamos uma estrutura de projeto e ferramentas as quais são suficientes para atingir o objetivo que propomos; de forma também a apresentar uma familiaridade aos desenvolvedores da versão atual do projeto.
Levando em conta todos os elementos que apresentamos, acreditamos que os pontos positivos se sobressaem em relação aos problemas listados. Desta maneira reunímos aqui considerações que detém valor aos desenvolvedores da aplicação em questão.

7 CONSIDERAÇÕES FINAIS

Nesta seção encerramos nosso trabalho recapitulando o que definimos como objetivo e a maneira que escolhemos para alcança-los.

7.1 CONCLUSÕES PRIMÁRIAS

Destacamos a seguir alguns pontos que resumem nossas contribuições e ressaltam detalhes importantes que devem ser considerados em uma possível continuação deste trabalho. Começamos comparando o Portal do Aluno com algumas características marcantes que observamos nos *softwares* voltados para a gestão de recursos. Através deste comparativo ressaltamos uma melhoria a ser buscada no Portal ao que tange sua capacidade de integração e acesso de suas funcionalidades. Citamos alguns casos onde esta melhoria de acesso seria benéfica para a comunidade que interage com o SIGA. Demarcamos este ponto como um objetivo a ser buscado e propomos uma série de passos para alcançá-lo. Escolhemos o uso de elementos da arquitetura REST como uma forma de ampliar o acesso aos componentes do Portal e também de abordar a dependência que seus módulos tem entre sí.
Implementamos um exemplo que representa uma primeira versão do *backend* de um dos módulos do Portal adotando elementos da arquitetura REST. Nele usamos um conjunto diferente de ferramentas. Substituímos o principal arcabouço utilizado em sua construção por outro com capacidades similares. Detalhamos como as novas ferramentas possibilitam simplificar tarefas comuns que compõe o desenvolvimento do Portal do Aluno. Tarefas como por exemplo a declaração de consultas SQL e a restrição de perfis de usuário para interagir com determinados recursos da aplicação. No exemplo que criamos tanto navegadores usados atualmente quanto outros clientes podem acessar as funcionalidades que expomos do Portal. Isto é possível desde que eles sejam capazes de se comunicar através do protocolo HTTP e utilizar o formato JSON para receber e enviar dados.

7.2 TRABALHOS FUTUROS

Aqui ressaltamos elementos que podem ser extendidos em uma continuação do que agregamos neste trabalho, tornando-o mais próximo de algo que possa ser utilizado pelos usuários do SIGA futuramente. Lembramos que o foco de nossa abordagem é o *backend* do Portal do Aluno. A composição da sua interface gráfica utilizando os artefatos que aqui apresentamos seria um trabalho à parte. Demarcamos a seguir algumas frentes que julgamos de interesse em uma continuação do que apresentamos até então.

7.2.1 Interação e adequação aos outros componentes do SIGA

Na aplicação que criamos qualquer usuário ou componente que necessitar dos serviços de um módulo do Portal pode acessá-los via HTTP. A única restrição para que tal acesso seja bem sucedido é que as requisições feitas devem conter o token de acesso com as devidas permissões (contidas no campo *roles* no caso do JWT) para o recurso requisitado. Neste trabalho validamos a adequação da nossa implementação onde os casos de uso envolvem apenas um módulo isolado. Posteriormente outros módulos do Portal do Aluno podem ser implementados seguindo os passos que determinamos. Tornando possível testar os casos onde o serviço de um dado módulo consulte dados que competem a um outro módulo distinto. Não identificamos motivos pelos quais nossa abordagem se limitaria apenas ao Portal do Aluno. Sendo assim outros sistemas que compõe o SIGA também podem ser implementados conforme o exemplo que fornecemos, tornando possível julgar melhor a adequação dos passos que determinamos.
Um dos cenários citados que se beneficiaria de nossa implementação seria relativo aos aplicativos *mobile*. Um ponto que merece ser avaliado posteriormente sobre estes aplicativos seria sobre a adequação de nossa implementação relativa às necessidades particulares destes artefatos. Principalmente no quesito de dados trafegados. Já que os dispositivos onde estas versões são utilizadas possuem uma sensibilidade maior com relação a este ponto. Devido ao fato de estarem sujeitos à limitação da franquia dos planos de dados. Em uma extensão deste trabalho pode ser avaliado este consumo e também propor formas de mitigar o consumo destes dados caso necessário.

7.2.2 Testes de integração em nossa proposta

Em nosso exemplo de implementação criamos testes automatizados para garantir o atendimento dos casos de uso mais simples de nosso artefato. O Portal do Aluno também possui sua própria suíte de testes porém utilizando ferramentas diferentes e os implementando de maneira distinta de como fizemos em nosso exemplo. No caso do Portal as dependências referentes à funcionalidades de outros módulos do Portal são empacotadas na própria aplicação que delas dependem, conforme explicamos nos capítulos anteriores. Em nosso caso estas dependências não estão presentes na própria aplicação, pois não as empacotamos mais no próprio código como antes. Elas precisam ser acessadas de outra forma. Nesse caso o serviço ou dado a ser consultado deve ser acessado usando o seu *endpoint* específico. No cenário que apresentamos, por causa do isolamento do módulo Avaliação com relação aos outros serviços do Portal, não foi necessário consultar outros módulos.
Existem casos no Portal contendo serviços mais complexos onde a interação com outros módulos do sistema é necessária. Os testes automatizados que irão cobrir estes serviços podem ser abordados de duas formas diferentes. A primeira seria criar os chamados

mocks[1]. Neles definimos um objeto que representa a resposta que obteríamos ao consultar um outro serviço, podendo representar a resposta de um *endpoint* que consultamos para obter os insumos necessários ao funcionamento de um outro serviço. O lado ruim desta abordagem é o cuidado necessário ao mudar um serviço para que os *mocks* criados reflitam esta mudança depois quando necessário. Outra forma de abordar a necessidade de integração com outro serviços nestes testes é criar um ambiente onde a versão desejada de cada módulo contendo eles está disponível para ser acessada. Apesar de ser uma opção mais custosa não haveria a necessidade de identificar e atualizar cada um dos *mocks* criados, focando apenas na versão do módulo que se deseja utilizar para os testes de integração do serviço em questão.

7.2.3 Mecanismos de restrição de escopo de acesso

O mecanismo de segurança que implementamos para nossa API não possui algumas das capacidades que o Portal do Aluno necessita para funcionar plenamente. Em nosso exemplo nos limitamos a demarcar os acessos aos *endpoints* da API de acordo com um determinado perfil de usuário que deve poder acessá-lo. Porém existem casos onde é necessário um controle mais fino do acesso aos dados. Como acontece na filtragem de conteúdo baseado no escopo do usuário estando sujeito a sua localização dentro da UFRJ (como secretarias de cursos distintos por exemplo). Podemos utilizar neste caso a capacidade dos tokens JWT de guardar informações para simplificar a implementação deste filtro. O mecanismo de autenticação e autorização que construímos para este exemplo poderia ainda ser aproveitado por outros constituintes do SIGA, como os sistemas Pré-Matrícula e SGA.

[1] <https://martinfowler.com/articles/mocksArentStubs.html>

REFERÊNCIAS

BELL, G. **Programmed Data Processor-1 Manual**. 1961. Disponível em: <https://gordonbell.azurewebsites.net/Digital/PDP%201%20Manual%201961.pdf>. Acesso em: 29 jan. 2020.

FIELDING, R. T. Architectural styles and the design of network-based software architectures. 2000. **University of California, Irvine**, p. 162, 2000.

FÁVERO, M. de Lourdes de A. A universidade do brasil um itinerário marcado de lutas. **Revista Brasileira de Educação**, n. 10, p. 1, 1999.

HURBEAN, L.; FOTACHE, D. Erp iii the promise of a new generation. In: ASE BUCHAREST. **Proceeding of the 13th International Conference on Informatics in Economy**. Bucharest, Romania, 2014. Proceeding of the 13th International Conference on Informatics in Economy, p. 3.

IBM. **1440 Data Processing System**. 2020. Disponível em: <https://www.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP1440.html>. Acesso em: 29 jan. 2020.

LAURINDO, F. J. B.; MESQUITA, M. A. de. Material requirements planning: 25 anos de história - uma revisão do passado e prospecção do futuro. **Gestão & Produção**, SciELO Brasil, v. 7, n. 3, p. 320–337, 2000.

MAJEED, A.; RAUF, I. Mvc architecture: A detailed insight to the modern web applications development. **Peer Review Journal of Solar & Photoenergy Systems**, v. 1, n. 1, 2018.

MCGAUGHEY, R. E.; GUNASEKARAN, A. Enterprise resource planning (erp): past, present and future. **International Journal of Enterprise Information Systems (IJEIS)**, IGI Global, v. 3, n. 3, p. 23–35, 2007.

MVC. 2017. Disponível em: <https://upload.wikimedia.org/wikipedia/commons/9/9d/MVC-basic.svg>. Acesso em: 25 mar. 2022.

NIST. 2022. Disponível em: <https://csrc.nist.gov/glossary/term/information_system>. Acesso em: 20 dez. 2022.

O'NEIL, E. J. Object/relational mapping 2008: Hibernate and the entity data model (edm). In: **Proceedings of the 2008 ACM SIGMOD international conference on Management of data**. [S.l.: s.n.], 2008. p. 1351–1356.

RASHID, M. A.; HOSSAIN, L.; PATRICK, J. D. The evolution of erp systems: A historical perspective. In: **Enterprise Resource Planning: Solutions and Management**. Rolla, Missouri: IGI global, 2002. p. 35–50.

RENS, P. J.; KLEIJNEN, J. P. C. Impact revisited: A critical analysis of ibm's inventory package "impact". **Production and Inventory Management**, APICS - The Educational Society for Resource Management, v. 19, n. 1, p. 71–90, 1978.

SACKMAN, H. Time-sharing versus batch processing: The experimental evidence. In: **Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference**. New York, NY, USA: Association for Computing Machinery, 1968. (AFIPS '68 (Spring)), p. 1–10. ISBN 9781450378970. Disponível em: <https://doi.org/10.1145/1468075.1468077>.

STORINO, R. Portal aluno da universidade federal do rio de janeiro: Um estudo avaliativo na perspectiva de graduandos. 2017.

WIGHT, O. **Manufacturing resource planning: MRP II: unlocking America's productivity potential**. [S.l.]: John Wiley & Sons, 1995.

WORTMANN, J. C. Evolution of erp systems. In: **Strategic Management of the manufacturing value chain**. [S.l.]: Springer, 1998. p. 11–23.

ZHANG, L.-J.; ZHANG, J.; CAI, H. Service-oriented architecture. **Services Computing**, Springer, p. 89–113, 2007.

APÊNDICES

ANEXOS

ANEXO A – NOSSA VERSÃO DA ENTIDADE AVALIAÇÃO

Código A.1 – Nossa versão da entidade Avaliação

```
package br.ufrj.siga.avaliacao;

import java.util.List;

import javax.persistence.AttributeOverride;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.validation.constraints.NotNull;

import org.apache.commons.lang3.builder.EqualsBuilder;
import org.apache.commons.lang3.builder.HashCodeBuilder;
import org.hibernate.annotations.Cascade;
import org.hibernate.annotations.CascadeType;
import org.hibernate.annotations.OrderBy;
import org.hibernate.validator.constraints.Length;

import com.fasterxml.jackson.annotation.JsonIdentityInfo;
import com.fasterxml.jackson.annotation.ObjectIdGenerators;

import br.ufrj.siga.ensino.SituacaoEntidade;
import br.ufrj.siga.ensino.TipoNivel;
import br.ufrj.siga.model.AbstractEntity;
import br.ufrj.siga.org.Segmentacao;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Entity
@AttributeOverride(name = "oid", column = @Column(name = "avaliacao_oid"))
@AllArgsConstructor @NoArgsConstructor
@JsonIdentityInfo(generator = ObjectIdGenerators.PropertyGenerator.class, property = "
    ↪ oid")
public class Avaliacao extends AbstractEntity
{
  @Getter @Setter
  @Length(max = 50)
```

```
    private String              nome;

    @Getter @Setter
    @Length(max = 255)
    private String              descricao;

    @Getter @Setter
    @Length(max = 2000)
    private String              instrucoes;

    @Getter @Setter
    @ManyToOne
    @JoinColumn(name = "tipoNivel_oid")
    private TipoNivel           tipoNivel;

    @Getter @Setter
    @ManyToOne
    @JoinColumn(name = "segmentacao_oid")
    @NotNull
    private Segmentacao         segmentacao;

    @Getter @Setter
    @ManyToOne
    @JoinColumn(name = "situacaoEntidade_oid")
    @NotNull
    private SituacaoEntidade    situacaoEntidade;

    @Getter @Setter
    @ManyToOne
    @JoinColumn(name = "tipoAvaliacao_oid")
    private TipoAvaliacao       tipoAvaliacao;

    @Getter @Setter
    @OneToMany(mappedBy = "avaliacao")
    @OrderBy(clause = "numOrdem asc")
    @Cascade(CascadeType.PERSIST)
    private List<GrupoAvaliacaoPergunta> gruposAvaliacaoPergunta;

    public boolean isAtiva()
    {
      return situacaoEntidade.isAtivo();
    }

    public boolean isDesativada()
    {
      return situacaoEntidade.isDesativado();
    }
```

```
  public boolean isNova()
  {
    return situacaoEntidade.isNovo();
  }

  public boolean isEmDesativacao()
  {
    return situacaoEntidade.isEmDesativacao();
  }

  public boolean isBasica()
  {
    return situacaoEntidade.isBasico();
  }

  @Override
  public int hashCode()
  {
    return new HashCodeBuilder(5, 11).append(oid).append(segmentacao).append(
        ↪ situacaoEntidade).toHashCode();
  }

  @Override
  public boolean equals(final Object avaliacao)
  {
    if (this == avaliacao)
    {
      return true;
    }
    if (!(avaliacao instanceof Avaliacao))
    {
      return false;
    }

    final Avaliacao other = (Avaliacao) avaliacao;

    return new EqualsBuilder().append(oid, other.oid)
                                          .append(segmentacao, other.segmentacao)
                              .append(situacaoEntidade, other.situacaoEntidade)
                              .isEquals();
  }
}
```

ANEXO B – VERSÃO ORIGINAL DA ENTIDADE AVALIACAO.JAVA

Código B.1 – Versão original da entidade Avaliação

```
package br.ufrj.gnosys.avaliacao;

import java.util.List;

import javax.persistence.Entity;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;

import org.apache.commons.lang.builder.EqualsBuilder;
import org.apache.commons.lang.builder.HashCodeBuilder;
import org.hibernate.validator.Length;
import org.hibernate.validator.NotNull;

import br.ufrj.gnosys.ensino.SituacaoEntidade;
import br.ufrj.gnosys.ensino.TipoNivel;
import br.ufrj.gnosys.model.AbstractEntity;
import br.ufrj.gnosys.org.Segmentacao;

@Entity
public class Avaliacao extends AbstractEntity
{

  private static final long    serialVersionUID = -1359813904304619357L;

  @Length(max = 50)
  private String               nome;

  @Length(max = 255)
  private String               descricao;

  @Length(max = 2000)
  private String               instrucoes;

  @ManyToOne
  private TipoNivel            tipoNivel;

  @ManyToOne
  @NotNull
  private Segmentacao          segmentacao;

  @NotNull
```

```
@ManyToOne
private SituacaoEntidade     situacaoEntidade;

@ManyToOne
private TipoAvaliacao        tipoAvaliacao;

@OneToMany
private List<GrupoAvaliacaoPergunta> gruposAvaliacaoPergunta;

public String getNome()
{
  return nome;
}

public void setNome(final String paramNome)
{
  nome = paramNome;
}

public String getDescricao()
{
  return descricao;
}

public void setDescricao(final String paramDescricao)
{
  descricao = paramDescricao;
}

public SituacaoEntidade getSituacaoEntidade()
{
  return situacaoEntidade;
}

public void setSituacaoEntidade(final SituacaoEntidade paramSituacaoEntidade)
{
  situacaoEntidade = paramSituacaoEntidade;
}

public Segmentacao getSegmentacao()
{
  return segmentacao;
}

public void setSegmentacao(final Segmentacao paramSegmentacao)
{
  segmentacao = paramSegmentacao;
```

```
}

public List<GrupoAvaliacaoPergunta> getGruposAvaliacaoPergunta()
{
  return gruposAvaliacaoPergunta;
}

public void setGruposAvaliacaoPergunta(List<GrupoAvaliacaoPergunta>
    ↪ gruposAvaliacaoPergunta)
{
  this.gruposAvaliacaoPergunta = gruposAvaliacaoPergunta;
}

public boolean isAtiva()
{
  return situacaoEntidade.isAtivo();
}

public boolean isDesativada()
{
  return situacaoEntidade.isDesativado();
}

public boolean isNova()
{
  return situacaoEntidade.isNovo();
}

public boolean isEmDesativacao()
{
  return situacaoEntidade.isEmDesativacao();
}

public boolean isBasica()
{
  return situacaoEntidade.isBasico();
}

public TipoNivel getTipoNivel()
{
  return tipoNivel;
}

public void setTipoNivel(final TipoNivel paramTipoNivel)
{
  tipoNivel = paramTipoNivel;
}
```

```
public TipoAvaliacao getTipoAvaliacao()
{
  return tipoAvaliacao;
}

public void setTipoAvaliacao(TipoAvaliacao tipoAvaliacao)
{
  this.tipoAvaliacao = tipoAvaliacao;
}

public String getInstrucoes()
{
  return instrucoes;
}

public void setInstrucoes(String instrucoes)
{
  this.instrucoes = instrucoes;
}

@Override
public int hashCode()
{
  return new HashCodeBuilder(5, 11).append(oid).append(segmentacao).append(
      ↪ situacaoEntidade).toHashCode();
}

@Override
public boolean equals(final Object avaliacao)
{
  if (this == avaliacao)
  {
    return true;
  }
  if (!(avaliacao instanceof Avaliacao))
  {
    return false;
  }

  final Avaliacao other = (Avaliacao) avaliacao;

  return new EqualsBuilder().append(oid, other.getOid()).append(segmentacao, other.
      ↪ getSegmentacao())
                        .append(situacaoEntidade, other.getSituacaoEntidade()).
                            ↪ isEquals();
}
```

```
}
```

ANEXO C – ARQUIVO DE MAPEAMENTO AVALIACAO.ORM.XML

Código C.1 – Mapeamento de relacionamentos da entidade Avaliação

```
<?xml version="1.0" encoding="utf-8"?>
<entity-mappings version="1.0"
        xmlns="http://java.sun.com/xml/ns/persistence/orm" xmlns:xsi="http:
            //www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
            http://java.sun.com/xml/ns/persistence/orm_1_0.xsd">

  <!--Queries-->
  <named-query name="
      qryRecuperarAvaliacoesPorCodigoTipoAvaliacaoECodigoSituacaoEntidade">
    <query>
      select A
      from Avaliacao A
      join A.situacaoEntidade SE
      join A.tipoAvaliacao TA
      where
        TA.codigo = :codigoTipoAvaliacao and
        SE.codigo = :codigoSituacaoEntidade
    </query>
  </named-query>

  <named-query name="
      qryRecuperarAlunosComAvaliacaoDisciplinaPendentePorPessoaFisica">
    <query>
      select
        a
      from
        Aluno a
        JOIN a.curso c
        JOIN FETCH a.situacaoMatricula sm
        JOIN FETCH a.pessoaFisica pf
      where
        exists(select
            1
          from
            RespostaAvaliacaoDisciplina rad
            JOIN rad.avaliacaoPergunta ap
            JOIN ap.grupoAvaliacaoPergunta gap
            JOIN gap.avaliacao av
            JOIN av.situacaoEntidade s
          where
```

```
            s.codigo = 'A' and
            rad.aluno = a

          ) and
        pf = :pessoaFisica
    order by
      a.matriculaDRE desc
  </query>
</named-query>

      <named-query name="qryRecuperarAvaliacaoDeCarga">
      <query>
      select distinct avaliacao
      from Avaliacao avaliacao
      where exists (
              SELECT rad
                      FROM RespostaAvaliacaoDisciplina rad
                      join rad.avaliacaoPergunta ap
          join ap.grupoAvaliacaoPergunta gap
                      WHERE gap.avaliacao = avaliacao
              )
      and avaliacao = :paramAvaliacao
      </query>
  </named-query>

      <named-query name="qryRecuperarAvaliacoesPorSituacaoEntidade">
              <query>
              select a
              from Avaliacao a
              join a.situacaoEntidade se
              where se.codigo = :codigo
              order by a.segmentacao.ano DESC, a.segmentacao.periodo DESC
  </query>
      </named-query>

      <named-query name="qryRecuperarAvaliacoes">
        <query>
        select a
        from Avaliacao a
        order by a.segmentacao.ano DESC, a.segmentacao.periodo DESC
        </query>
      </named-query>

<named-query name="qryRecuperarAvaliacoesAnteriores">
  <query>
      <![CDATA[
      select a from Avaliacao a
```

```
        where a.segmentacao.ano <= :ano
        order by a.segmentacao.ano DESC, a.segmentacao.periodo DESC
        ]]>
    </query>
</named-query>

<named-query name="qryRecuperarAvaliacoesAnterioresPorTipo">
    <query>
        <![CDATA[
        select a from Avaliacao a
        where a.segmentacao.ano <= :ano and a.tipoAvaliacao = :tipo
        order by a.segmentacao.ano DESC, a.segmentacao.periodo DESC
        ]]>
    </query>
</named-query>

<named-query name="qryRecuperarAvaliacaoPorSegmentacao">
        <query>
        select distinct avaliacao
        from Avaliacao avaliacao
        where avaliacao.segmentacao = :segmentacao
        </query>
</named-query>

<named-query name="qryRecuperarAvaliacaoPorSegmentacaoEPorTipoAvaliacao">
        <query>
        select distinct avaliacao
        from Avaliacao avaliacao
        where avaliacao.segmentacao = :segmentacao and avaliacao.
            tipoAvaliacao = :tipoAvaliacao
        </query>
</named-query>

<named-query name="qryRecuperarTodasAvaliacoesDisciplina">
        <query>
        select avaliacao
        from Avaliacao avaliacao
        where avaliacao.tipoAvaliacao.codigo = '01'
        order by avaliacao.segmentacao.ano DESC, avaliacao.segmentacao.
            periodo DESC
        </query>
</named-query>

<!--Mapeamentos da Entidade-->
<entity class="br.ufrj.gnosys.avaliacao.Avaliacao">
                <attribute-override name="oid">
                        <column name="Avaliacao_oid" />
```

```
                </attribute-override>
                <attributes>
                        <many-to-one name="segmentacao" >
                                <join-column name="segmentacao_oid"  />
                        </many-to-one>
                        <many-to-one name="situacaoEntidade">
                                <join-column name="situacaoEntidade_oid" />
                        </many-to-one>
                        <many-to-one name="tipoNivel">
                                <join-column name="tipoNivel_oid" />
                        </many-to-one>
        <many-to-one name="tipoAvaliacao">
          <join-column name="tipoAvaliacao_oid" />
        </many-to-one>
                        <one-to-many name="gruposAvaliacaoPergunta" mapped-
                            by="avaliacao">
                                <order-by>numOrdem asc</order-by>
                                <cascade><cascade-persist/></cascade>
                        </one-to-many>

                </attributes>
        </entity>
</entity-mappings>
```

ANEXO D – EXEMPLO DE DAO DO PORTAL DO ALUNO

Código D.1 – DAO da entidade Avaliação

```
package br.ufrj.gnosys.avaliacao;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.List;

import javax.persistence.Query;

import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.AutoCreate;
import org.jboss.seam.annotations.Factory;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Scope;

import br.ufrj.gnosys.avaliacao.controle.AvaliacaoDisciplinaVO;
import br.ufrj.gnosys.avaliacao.controle.AvaliacaoTurmaVO;
import br.ufrj.gnosys.avaliacao.controle.AvaliacaoVO;
import br.ufrj.gnosys.framework.system.Usuario;
import br.ufrj.gnosys.model.AbstractDAO;
import br.ufrj.gnosys.org.Segmentacao;
import br.ufrj.gnosys.pessoa.PessoaFisica;
import br.ufrj.gnosys.registro.Aluno;
import br.ufrj.gnosys.util.SQLScriptReader;

@Name("avaliacaoDAO")
@AutoCreate
@Scope(ScopeType.CONVERSATION)
public class AvaliacaoDAO extends AbstractDAO<Avaliacao>
{

  private static final long serialVersionUID = 5329735499795818945L;

  public void carregarRespostasAvaliacaoDisciplina(final String segmentacaoAno, final
      ↪ String segmentacaoPeriodo)
  {
    String query = SQLScriptReader.read(getClass().getResourceAsStream("
        ↪ qryCarregarRespostasAvaliacaoDisciplina.sql"));

    final Query q = getEntityManager().createNativeQuery(query);

    q.setParameter("ano", segmentacaoAno);
```

```
  q.setParameter("periodo", segmentacaoPeriodo);
                        q.executeUpdate();
}

public void carregarRespostasAvaliacaoTurma(final String segmentacaoAno, final
    ↪ String segmentacaoPeriodo,
                                        final String tipoNivelCodigo)
{
  String query = SQLScriptReader.read(getClass().getResourceAsStream("
      ↪ qryCarregarRespostasAvaliacaoTurma.sql"));

  final Query q = getEntityManager().createNativeQuery(query);

  q.setParameter("ano", segmentacaoAno);
  q.setParameter("periodo", segmentacaoPeriodo);
  q.setParameter("codigo", tipoNivelCodigo);
  q.executeUpdate();
}

@SuppressWarnings("unchecked")
public List<Avaliacao> recuperarTodasAsAvaliacoes()
{
  final List<Avaliacao> avaliacoes = getEntityManager().createNamedQuery("
      ↪ qryRecuperarAvaliacoes").getResultList();

  return avaliacoes;
}

@SuppressWarnings("unchecked")
public List<Avaliacao> recuperarTop5Avaliacoes()
{
  final List<Avaliacao> avaliacoes = getEntityManager().createNamedQuery("
      ↪ qryRecuperarAvaliacoes").setMaxResults(5).getResultList();

  return avaliacoes;
}

@SuppressWarnings("unchecked")
public List<AvaliacaoPergunta> recuperaAvaliacaoPerguntas(final Avaliacao avaliacao)
{

  final List<AvaliacaoPergunta> listAvPerguntas = getEntityManager().createNamedQuery
      ↪ ("qryRecuperarAvaliacaoPerguntaByAvaliacao").setParameter("avaliacao",
      ↪ avaliacao).getResultList();

  return listAvPerguntas;
```

```
}

@SuppressWarnings("unchecked")
public List<Avaliacao> recuperarAvaliacoesDisponiveisPorUsuario(Usuario usuario)
{
  final List<Avaliacao> listaAvaliacoesDisponiveis;

  listaAvaliacoesDisponiveis = getEntityManager().createNamedQuery("
      ↪ qryRecuperarAvaliacoesDisponiveisPorUsuario").setParameter("usuario",
      ↪ usuario).getResultList();

  return listaAvaliacoesDisponiveis;
}

@SuppressWarnings("unchecked")
public Avaliacao recuperarAvaliacaoPorSegmentacao(final Segmentacao segmentacao)
{
  final List<Avaliacao> listaAvaliacoes;

  listaAvaliacoes = getEntityManager().createNamedQuery("
      ↪ qryRecuperarAvaliacaoPorSegmentacao").setParameter("segmentacao",
      ↪ segmentacao).getResultList();

  if (listaAvaliacoes.isEmpty())
  {
    return null;
  }
  else
  {
    return listaAvaliacoes.get(0);
  }
}

@SuppressWarnings("unchecked")
public Avaliacao recuperarAvaliacaoPorSegmentacaoEPorTipoAvaliacao(Segmentacao
    ↪ segmentacao,TipoAvaliacao tipoAvaliacao)
{
  final List<Avaliacao> listaAvaliacoes;

  listaAvaliacoes = getEntityManager().createNamedQuery("
      ↪ qryRecuperarAvaliacaoPorSegmentacaoEPorTipoAvaliacao").setParameter("
      ↪ segmentacao", segmentacao).setParameter("tipoAvaliacao", tipoAvaliacao).
      ↪ getResultList();

  if (listaAvaliacoes.isEmpty())
  {
    return null;
```

```
  }
  else
  {
    return listaAvaliacoes.get(0);
  }
}

@SuppressWarnings("unchecked")
public List<Aluno> recuperarAlunosComAvaliacaoPendentePorPessoaFisica(final
    ↪ PessoaFisica pessoaFisica)
{

  final List<Aluno> alunosDaPessoaFisica = getEntityManager().createNamedQuery("
      ↪ qryRecuperarAlunosComAvaliacaoDisciplinaPendentePorPessoaFisica").
      ↪ setParameter("pessoaFisica", pessoaFisica).getResultList();

  return alunosDaPessoaFisica;
}

@SuppressWarnings("unchecked")
public boolean jaExisteAvaliacaoDadaSegmentacao(final Segmentacao segmentacao)
{
  final List<Avaliacao> list = getEntityManager().createNamedQuery("
      ↪ qryRecuperarAvaliacaoPorSegmentacao").setParameter("segmentacao",
      ↪ segmentacao).getResultList();

  return !list.isEmpty();
}

@SuppressWarnings("unchecked")
public List<Avaliacao> recuperarAvaliacaoPorSituacaoEntidade(final String
    ↪ situacaoEntidadeCodigo)
{
  return getEntityManager().createNamedQuery("
      ↪ qryRecuperarAvaliacoesPorSituacaoEntidade").setParameter("codigo",
      ↪ situacaoEntidadeCodigo).getResultList();
}

@Override
public Avaliacao getInstance()
{
  return super.getInstance();
}

@SuppressWarnings("unchecked")
public boolean detectarCargaDeAvaliacao(final Avaliacao avaliacao)
{
```

```
  List<Avaliacao> resultList = getEntityManager().createNamedQuery("
      ↪ qryRecuperarAvaliacaoDeCarga").setParameter("paramAvaliacao", avaliacao).
      ↪ getResultList();

  return !resultList.isEmpty();
}

@SuppressWarnings("unchecked")
public List<Avaliacao> recuperarAvaliacoesAnteriores(String ano)
{
  List<Avaliacao> avaliacoesAnteriores = getEntityManager().createNamedQuery("
      ↪ qryRecuperarAvaliacoesAnteriores").setParameter("ano", ano).getResultList();

  return avaliacoesAnteriores;
}

@SuppressWarnings("unchecked")
public List<Avaliacao> recuperarAvaliacoesAnterioresPorTipo(String ano,
    ↪ TipoAvaliacao tipo)
{
  List<Avaliacao> avaliacoesAnteriores = getEntityManager().createNamedQuery("
      ↪ qryRecuperarAvaliacoesAnterioresPorTipo").setParameter("ano", ano).
      ↪ setParameter("tipo", tipo).getResultList();

  return avaliacoesAnteriores;
}

public void persistirNovaAvaliacao(Avaliacao novaAvaliacao)
{
  getEntityManager().persist(novaAvaliacao);
  getEntityManager().flush();
}

public void popularEstatisticaAvaliacaoDisciplina(Avaliacao avaliacao)
{
  String query = SQLScriptReader.read(getClass().getResourceAsStream("controle/sql/
      ↪ qryPopularEstatisticaAvaliacaoDisciplina.sql"));

  getEntityManager().createNativeQuery(query).setParameter("anoAvaliacao", avaliacao.
      ↪ getSegmentacao().getAno()).setParameter("periodoAvaliacao", avaliacao.
      ↪ getSegmentacao().getPeriodo()).executeUpdate();
}

@SuppressWarnings("unchecked")
public List<AvaliacaoVO> recuperarAvaliacoesDisciplinaVOPorAvaliacao(Avaliacao
    ↪ avaliacao)
{
```

```
  String query = SQLScriptReader.read(getClass().getResourceAsStream("controle/sql/
      ↪ qryParcialAvaliacaoDisciplina.sql"));

  List<Object[]> objetos = getEntityManager().createNativeQuery(query).setParameter("
      ↪ anoAvaliacao", avaliacao.getSegmentacao().getAno()).setParameter("
      ↪ periodoAvaliacao", avaliacao.getSegmentacao().getPeriodo()).getResultList();

  List<AvaliacaoVO> avaliacoes = mapearAvaliacoesDisciplinaVO(objetos);

  return avaliacoes;
}

@SuppressWarnings("unchecked")
public AvaliacaoVO recuperarAvaliacaoSistemaVOPorAvaliacao(Avaliacao avaliacao)
{
  String query = SQLScriptReader.read(getClass().getResourceAsStream("controle/sql/
      ↪ qryParcialAvaliacaoSistema.sql"));

  List<Object[]> objetos = getEntityManager().createNativeQuery(query).setParameter("
      ↪ avaliacaoOid", avaliacao.getOid()).getResultList();

  AvaliacaoVO avaliacaoSistemaVO = mapearAvaliacaoSistemaVO(objetos);

  return avaliacaoSistemaVO;
}

private List<AvaliacaoVO> mapearAvaliacoesDisciplinaVO(List<Object[]> objetos)
{
  List<AvaliacaoVO> avaliacoes = new ArrayList<AvaliacaoVO>();

  for (Object[] obj : objetos)
  {
    avaliacoes.add(new AvaliacaoDisciplinaVO((String) obj[0], (Integer) obj[1], (
        ↪ Integer) obj[2], (BigDecimal) obj[3]));
  }

  return avaliacoes;
}

public void popularEstatisticaAvaliacaoTurma(Avaliacao avaliacao)
{
  String query = SQLScriptReader.read(getClass().getResourceAsStream("controle/sql/
      ↪ qryPopularEstatisticaAvaliacaoTurma.sql"));

  getEntityManager().createNativeQuery(query).setParameter("anoAvaliacao", avaliacao.
      ↪ getSegmentacao().getAno()).setParameter("periodoAvaliacao", avaliacao.
      ↪ getSegmentacao().getPeriodo()).executeUpdate();
```

```
}

@SuppressWarnings("unchecked")
public List<AvaliacaoVO> recuperarAvaliacoesTurmaVOPorAvaliacao(Avaliacao avaliacao)
{
  String query = SQLScriptReader.read(getClass().getResourceAsStream("controle/sql/
      ↪ qryParcialAvaliacaoTurma.sql"));

  List<Object[]> objetos = getEntityManager().createNativeQuery(query).setParameter("
      ↪ anoAvaliacao", avaliacao.getSegmentacao().getAno()).setParameter("
      ↪ periodoAvaliacao", avaliacao.getSegmentacao().getPeriodo()).getResultList();

  List<AvaliacaoVO> avaliacoes = mapearAvaliacoesTurmaVO(objetos);

  return avaliacoes;
}

private List<AvaliacaoVO> mapearAvaliacoesTurmaVO(List<Object[]> objetos)
{
  List<AvaliacaoVO> avaliacoes = new ArrayList<AvaliacaoVO>();

  for (Object[] obj : objetos)
        {
    avaliacoes.add(new AvaliacaoTurmaVO((String) obj[0], (String) obj[1], (Integer)
        ↪ obj[2], (Integer) obj[3], (BigDecimal) obj[4]));
  }

  return avaliacoes;
}

private AvaliacaoVO mapearAvaliacaoSistemaVO(List<Object[]> objetos)
{
  Object[] objeto = objetos.get(0);

  AvaliacaoVO avaliacaoVO = new AvaliacaoVO((Integer) objeto[0], (Integer) objeto[1],
      ↪  (BigDecimal) objeto[2]);

  return avaliacaoVO;
}

@SuppressWarnings("unchecked")
@Factory(value="avaliacoesDisciplina")
public List<Avaliacao> recuperarTodasAvaliacoesDisciplina()
{
  return getEntityManager().createNamedQuery("qryRecuperarTodasAvaliacoesDisciplina")
      ↪ .getResultList();
}
```

```
    @SuppressWarnings("unchecked")
    public List<Avaliacao>
        ↪ recuperarAvaliacoesPorCodigoTipoAvaliacaoECodigoSituacaoEntidade(final String
        ↪ codigoTipoAvaliacao, final String codigoSituacaoEntidade)
    {
      return getEntityManager().createNamedQuery("
          ↪ qryRecuperarAvaliacoesPorCodigoTipoAvaliacaoECodigoSituacaoEntidade").
          ↪ setParameter("codigoTipoAvaliacao", codigoTipoAvaliacao).setParameter("
          ↪ codigoSituacaoEntidade", codigoSituacaoEntidade).getResultList();
    }
  }
```

ANEXO E – EXEMPLO DE UM REPOSITÓRIO DO ARCABOUÇO *SPRING*

Código E.1 – *Repository* da entidade Avaliação

```
package br.ufrj.siga.avaliacao.repository;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import br.ufrj.siga.avaliacao.Avaliacao;

@Repository
public interface AvaliacaoRepository extends JpaRepository<Avaliacao, Long> {
    public List<Avaliacao> findAll();

    public List<Avaliacao> findAvaliacaoByTipoAvaliacao_codigo(String codigo);

    public Avaliacao findAvaliacaoByOid(String oid);

    @Query(value =
          "SELECT parcial.curso AS cursoNome, parcial.totalAvaliadores AS
             ↪ totalAvaliadores, parcial.totalRespostas AS totalRespostas, parcial.
             ↪ porcentagemRespostas AS porcentagemRespostas "
        + "FROM ( "
        + "SELECT curso1.nome AS 'curso', 0 AS 'totalAvaliadores', COUNT(aluno1.
            ↪ aluno_oid) AS 'totalRespostas', 0 AS 'porcentagemRespostas' "
              + "FROM Aluno aluno1 "
              + "INNER JOIN Curso curso1 ON curso1.curso_oid = aluno1.curso_oid "
              + "WHERE EXISTS ( "
                    + "SELECT 1 "
                    + "FROM RespostaAvaliacaoDisciplina resposta1 "
                    + "INNER JOIN AvaliacaoPergunta pergunta1 ON pergunta1.
                        ↪ avaliacaoPergunta_oid = resposta1.avaliacaoPergunta_oid "
                    + "INNER JOIN GrupoAvaliacaoPergunta grupo1 ON grupo1.
                        ↪ grupoAvaliacaoPergunta_oid = pergunta1.
                        ↪ grupoAvaliacaoPergunta_oid "
                    + "INNER JOIN Avaliacao avaliacao1 ON avaliacao1.avaliacao_oid =
                        ↪ grupo1.avaliacao_oid "
                    + "WHERE avaliacao1.avaliacao_oid = ?1 AND resposta1.valor IS NOT
                        ↪  NULL "
              + ") "
```

```
            + "GROUP BY curso1.nome "
            + "UNION "
            + "SELECT curso2.nome AS 'Curso', COUNT(aluno2.aluno_oid) AS 'Total
                ↪ Avaliadores', 0 AS 'Total Respostas', 0 AS 'porcentagemRespostas'
                ↪  "
            + "FROM Aluno aluno2 "
            + "INNER JOIN Curso curso2 ON curso2.curso_oid = aluno2.curso_oid "
            + "WHERE EXISTS ( "
                    + "SELECT 1 "
                    + "FROM RespostaAvaliacaoDisciplina resposta2 "
                    + "INNER JOIN AvaliacaoPergunta pergunta2 ON pergunta2.
                        ↪ avaliacaoPergunta_oid = resposta2.avaliacaoPergunta_oid "
                    + "INNER JOIN GrupoAvaliacaoPergunta grupo2 ON grupo2.
                        ↪ grupoAvaliacaoPergunta_oid = pergunta2.
                        ↪ grupoAvaliacaoPergunta_oid "
                    + "INNER JOIN Avaliacao avaliacao2 ON avaliacao2.avaliacao_oid =
                        ↪ grupo2.avaliacao_oid "
                    + "WHERE avaliacao2.avaliacao_oid = ?1 "
            + ") "
            + "GROUP BY curso2.nome "
            + ") AS Parcial"
            , nativeQuery = true)
    public List<ParcialAvaliacaoDTO> recuperarParcialAvaliacaoDisciplinaPorAvaliacao(
        ↪ @Param(value = "avaliacao_oid") String avaliacao_oid);
}
```

ANEXO F – CONTROLADOR RELACIONADO À ENTIDADE AVALIAÇÃO

Código F.1 – Controlador da entidade Avaliação

```
package br.ufrj.siga.avaliacao.api;

import java.io.IOException;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.bind.annotation.RestController;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;

import br.ufrj.siga.avaliacao.cadastro.CadastroAvaliacaoBean;
import br.ufrj.siga.avaliacao.repository.AvaliacaoRepository;

@CrossOrigin
@RestController
@RequestMapping("/Avaliacao")
public class AvaliacaoController {

        @Autowired
        private AvaliacaoRepository avaliacaoRepository;

        @Autowired
        private CadastroAvaliacaoBean cadastroAvaliacaoBean;

        private ObjectMapper objectMapper = new ObjectMapper().configure(
            ↪ DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false);

        @ResponseBody
        @GetMapping(value = "/avaliacao", produces = MediaType.
            ↪ APPLICATION_JSON_UTF8_VALUE)
```

```
        @PreAuthorize("hasAuthority('Manutenção')")
        public String recuperarAvaliacao() throws IOException
        {
                return objectMapper.writeValueAsString(avaliacaoRepository.findAll());
        }

        @ResponseBody
        @GetMapping(value = "/avaliacao/{avaliacao_oid}", produces = MediaType.
            ↪ APPLICATION_JSON_UTF8_VALUE)
        @PreAuthorize("hasAuthority('Manutenção')")
        public String recuperarAvaliacao(@PathVariable String avaliacao_oid) throws
            ↪ JsonProcessingException
        {
                return objectMapper.writeValueAsString(avaliacaoRepository.
                    ↪ findAvaliacaoByOid(avaliacao_oid));
        }

        @ResponseBody
        @RequestMapping(value = "/avaliacao", method = RequestMethod.POST, consumes =
            ↪ MediaType.APPLICATION_JSON_UTF8_VALUE)
        @PreAuthorize("hasAuthority('Manutenção')")
        public ResponseEntity<Object> cadastrarAvaliacao(@RequestBody String jsonString
            ↪ )
        {
                return cadastroAvaliacaoBean.cadastrarAvaliacao(jsonString);
        }

        @ResponseBody
        @RequestMapping(value = "/avaliacao", method = RequestMethod.PATCH, consumes =
            ↪ MediaType.APPLICATION_JSON_UTF8_VALUE)
        @PreAuthorize("hasAuthority('Manutenção')")
        public ResponseEntity<Object> editarAvaliacao(@RequestBody String jsonString)
        {
                return cadastroAvaliacaoBean.cadastrarAvaliacao(jsonString);
        }
}
```

ANEXO G – CLASSE RELACIONADA AO CADASTRO DE AVALIAÇÕES

Código G.1 – Classe contendo a lógica de cadastro de avaliações

```
package br.ufrj.siga.avaliacao.cadastro;

import java.io.IOException;
import java.util.Date;
import java.util.UUID;

import javax.transaction.Transactional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Component;

import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;

import br.ufrj.siga.avaliacao.Avaliacao;
import br.ufrj.siga.avaliacao.ResponseUtils;
import br.ufrj.siga.avaliacao.TipoAvaliacao;
import br.ufrj.siga.avaliacao.repository.AvaliacaoRepository;
import br.ufrj.siga.avaliacao.repository.SituacaoEntidadeRepository;
import br.ufrj.siga.ensino.TipoNivel;
import br.ufrj.siga.org.Segmentacao;

@Component
public class CadastroAvaliacaoBean {

        @Autowired
        private AvaliacaoRepository avaliacaoRepository;

        @Autowired
        private SituacaoEntidadeRepository situacaoEntidadeRepository;

        private ObjectMapper objectMapper = new ObjectMapper().configure(
            ↪ DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false);

        @Transactional
        public ResponseEntity<Object> cadastrarAvaliacao(String jsonString)
        {
                if(jsonString.isEmpty())
                        return ResponseUtils.gerarResponse("Requisição com corpo vazio.",
```

```
                    ↪  HttpStatus.BAD_REQUEST);

        Avaliacao avaliacao;
        boolean isEdicao;

        try {
                avaliacao = objectMapper.readValue(jsonString, Avaliacao.class);
                isEdicao = avaliacao.getOid() != null;
        } catch (IOException e) {
                return ResponseUtils.gerarResponse("Erro ao parsear JSON.",
                    ↪ HttpStatus.BAD_REQUEST);
        }

        if(!validar(avaliacao))
                return ResponseUtils.gerarResponse("Um dos campos não está
                    ↪ devidamente preenchido: Tipo Nıivel, Tipo Avaliação ou
                    ↪ Segmentação", HttpStatus.BAD_REQUEST);

        try {
                return salvar(avaliacao, isEdicao);
        } catch(Exception e) {
                return ResponseUtils.gerarResponse("Um dos campos não está
                    ↪ devidamente preenchido: Tipo Nıivel, Tipo Avaliação ou
                    ↪ Segmentação", HttpStatus.BAD_REQUEST);
        }
}

private ResponseEntity<Object> salvar(Avaliacao avaliacao, boolean isEdicao)
{
        if(isEdicao)
        {
                avaliacaoRepository.saveAndFlush(avaliacao);
                return ResponseUtils.gerarResponse("Avaliação editada com sucesso
                    ↪ !", HttpStatus.OK);
        } else {
                avaliacao.setOid(UUID.randomUUID().toString());
                avaliacao.setAutorizado(false);
                avaliacao.setUltimaAtualizacao(new Date());
                avaliacao.setSituacaoEntidade(situacaoEntidadeRepository.
                    ↪ findSituacaoEntidadeByCodigo("N"));
                avaliacaoRepository.saveAndFlush(avaliacao);
                return ResponseUtils.gerarResponse("Avaliação cadastrada com
                    ↪ sucesso!", HttpStatus.OK);
        }
}

protected boolean validar(Avaliacao avaliacao)
```

```
{
        if( !validarSegmentacao(avaliacao.getSegmentacao()) || !validarTipoNivel
            ↪ (avaliacao.getTipoNivel()) || !validarTipoAvaliacao(avaliacao.
            ↪ getTipoAvaliacao()))
                return false;

        if(avaliacao.getNome() == null || avaliacao.getDescricao() == null ||
            ↪ avaliacao.getInstrucoes() == null)
                return false;

        return true;
}

private boolean validarSegmentacao(Segmentacao segmentacao)
{
        if(segmentacao == null ||
                segmentacao.getAno() == null        || !isNumeric(segmentacao.
                    ↪ getAno())  ||
                segmentacao.getPeriodo() == null    || !isNumeric(segmentacao.
                    ↪ getPeriodo()) ||
                segmentacao.getBloco() == null      || !isNumeric(segmentacao.
                    ↪ getBloco()))
                                return false;

        return true;
}

private boolean validarTipoNivel(TipoNivel tipoNivel)
{
        if(tipoNivel == null || tipoNivel.getCodigo() == null || tipoNivel.
            ↪ getDescricao() == null)
                return false;

        return true;
}

private boolean validarTipoAvaliacao(TipoAvaliacao tipoAvaliacao)
{
        if(tipoAvaliacao == null || tipoAvaliacao.getCodigo() == null ||
            ↪ tipoAvaliacao.getDescricao() == null)
                return false;

        return true;
}

private boolean isNumeric(String string)
{
```

```
            return string.matches("-?\\d+(\\.\\d+)?");
        }
    }
```