FACULDADE DE TECNOLOGIA DE SÃO BERNARDO DO CAMPO
"ADIB MOISÉS DIB"

CHARLES GONÇALVES DIAS
GABRIEL SILVA DE MENEZES
MARCO ANTONIO ROCHA
PEDRO HIGOR BASTOS WERNECK
RAFAEL DE SOUZA FERREIRA

SIGAPI: UMA ABORDAGEM REST PARA ACESSO AO SIGA

São Bernardo do Campo
Junho/2017

CHARLES GONÇALVES DIAS
GABRIEL SILVA DE MENEZES
MARCO ANTONIO ROCHA
PEDRO HIGOR BASTOS WERNECK
RAFAEL DE SOUZA FERREIRA

SIGAPI: UMA ABORDAGEM REST PARA ACESSO AO SIGA

Trabalho de conclusão de curso apresentado na Faculdade de Tecnologia de São Bernardo para obtenção do título de Tecnólogo em Informática para Negócios.

Orientadora: Prof. Ma. Jediane Teixeira de Souza
Co orientador: Prof. Esp. Douglas Duarte

São Bernardo do Campo
Junho/2017

CHARLES GONÇALVES DIAS
GABRIEL SILVA DE MENEZES
MARCO ANTONIO ROCHA
PEDRO HIGOR BASTOS WERNECK
RAFAEL DE SOUZA FERREIRA

SIGAPI: UMA ABORDAGEM REST PARA ACESSO AO SIGA

Trabalho de conclusão de curso apresentado na Faculdade de Tecnologia de São Bernardo “Adib Moisés Dib” como requisito parcial para obtenção do título de Tecnólogo em Informática para Negócios.

Trabalho de Conclusão de Curso apresentado e aprovado em: 29/05/2017

Banca Examinadora:

_____
Prof. Ma. Jediane Teixeira de Souza, FATEC SBC – Orientador

_____
Prof. Me. Darli Regina Paschoalini Vaccari, FATEC SBC – Avaliador

_____
Prof. Antônio Carlos de Alcantara Thimóteo, FATEC SBC – Avaliador

Dedicamos esse trabalho aos professores: Prof. Me. Antonio Carlos de Alcantara Thimóteo, Ma. Cristiane Gomes de Carvalho Fontana, Prof. Esp. Douglas Duarte e Ma. Jediane Teixeira de Souza, pelo acompanhamento, paciência e incentivo que fizeram realidade a finalização deste projeto.

Agradecemos aos professores e professoras da FATEC SBC pela dedicação ao longo deste caminho, a nossos familiares pelo apoio e paciência e a nossos colegas de turma.

“Não sobrecarreguem seus corações pensando no melhor caminho. Pode ser que as trilhas nas quais cada um de vocês deve pisar já estejam diante de seus pés, embora talvez não consigam enxergá-las”.

J. R. R. Tolkien

RESUMO

A Faculdade de Tecnologia de São Paulo (FATEC), atualmente, é composta por mais de 80 mil alunos e 3300 professores, espalhados em 66 campus. Nessa conjuntura, é necessário um sistema robusto para gerenciar as informações de todo esse corpo discente. O SIGA (Sistema Integrado de Gestão Acadêmica) foi a solução adotada pelo Centro Paula Souza, uma autarquia do Governo do Estado de São Paulo responsável pela administração da FATEC, para controlar todo esse ambiente. O sistema tem como algumas das atividades: o controle de presença; administração das notas; repositório de material de estudo e apoio; processo de rematrícula; etc. Com todo esse conteúdo, existem inúmeras possibilidades de aplicar esses elementos em outras programações, como: aplicativos, *websites* interativos, ferramentas de pesquisa, etc. Contudo, é necessário que se abra o caminho para futuros desenvolvedores. Essa carência impulsionou a vontade de elaborar um meio para facilitar os próximos programadores e trazer o conhecimento de novas tecnologias à comunidade. Utilizando a estrutura de interface chamada API (*Application Programming Interface*), esse projeto pretende apresentar instrumentos que facilitem a criação de novas ferramentas alimentadas com as informações do SIGA.

Palavras-chave: API (*Application Programming Interface*). REST. SIGA. (Sistema Integrado de Gestão Acadêmica). FATEC. Centro Paula Souza.

ABSTRACT

Faculdade de Tecnologia de São Paulo (FATEC), currently, is composed of more than 80 thousand students and 3300 professors, across 66 campuses. A robust system is needed to manage all the information from the student body in this juncture. Centro Paula Souza, an autarchy of São Paulo Government responsible for FATEC's management, adopted SIGA (Sistema Integrado de Gestão Acadêmica) as a solution to control the whole environment. The system, among other functionalities, allows: to control students' attendance; score management; repository of study and support material; re-enrolling process; etc. With all this content, there are numerous possibilities to apply these elements in other programmings, such as: applications, interactive websites, search tool, etc. However, it is necessary to open the door for future developers and technologies. This lack has fuelled a desire to devise a means to facilitate the next developers and bring knowledge of new technologies to the community. Using the interface structure called API (Application Programming Interface) this project intends to present tools that ease the creation of the new tools fed with SIGA information.

Key Words: API (Application Programming Interface). REST. SIGA (Sistema Integrado de Gestão Acadêmica). FATEC. Centro Paula Souza.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
FATEC	Faculdade de Tecnologia
GPL	General Public License
HTML	Hypertext Markup Language
HTTPs	Hyper Text Transfer Protocol Secure
IDE	Integrated Development Environment
IoC	Inversion of Control
JSP	Java Server Pages
JVM	Java Virtual Machine
JWT	Json Web Token
Java EE	Java Enterprise Edition
Java SE	Java Standard Edition
Json	JavaScript Object Notation
LGPL	Lesser General Public License
MIT	Massachusetts Institute of technology
REST	Representational State Transfer
RSS	Really Simple Syndication
SIGA	Sistema Integrado de Gestão Acadêmica
SOAP	Simple Object Access Protocol
SVC	Sistema de Controle de Versão
TDD	Test Driven Development
TI	Tecnologia da Informação
TIC	Tecnologia da Informação e Comunicação
URL	Uniform Resource Locator
WWW	World Wide Web
XML	Extensible Markup Language

SUMÁRIO

INTRODUÇÃO

O século XXI está extremamente globalizado. As barreiras que impediam a cooperação estão sendo vencidas. Língua, distância e cultura são decifradas, absorvidas e transformadas nesse século. A internet, que facilitou a comunicação entre pessoas de lugares distantes, está articulada por emissões, ondas, mensagens, signos, símbolos, redes e alianças que tecem os lugares e as atividades, os campos e as cidades, as diferenças e as identidades, as nações e as nacionalidades. A tecnologia fomentou esse processo e impulsionou a adaptação por meio da padronização. *Hardwares* de várias empresas distintas se comunicam com uma imensa gama de *softwares* diferentes, sistemas interagem entre si, dados trafegam de uma plataforma para outra. A sobrevivência de uma tecnologia, ou até mesmo de uma empresa, depende dessa abertura, dessa habilidade de integrar-se.

Da necessidade de integração nasceu um recurso que, atualmente, é muito encontrando em páginas da internet. Opções de *Like* ou *Share* no Facebook, botões para enviar mensagens via Twitter ou compartilhar algum conteúdo no LinkedIn, vídeos do Youtube e mapas do Google em vários locais da internet são possíveis graças a utilização de um recurso conhecido como API (Application Programming Interface, ou Interface de Programação de Aplicação, em português). Um pacote de instruções que podem ser manipuladas por desenvolvedores para utilizar recursos de outras plataformas, sistemas ou *websites*, permitindo ao desenvolvedor estender algumas funcionalidades de seu próprio sistema ou aplicação. Utilizando-se da API do Google Maps, por exemplo, um programador pode inserir a opção de criação de rotas até um estabelecimento, ou até mesmo mostrar condições do tempo em uma determinada data, aplicando a API do Google Weather – aplicação com informações meteorológicas.

Com toda essa interação, a comunicação entre sistemas se tornou primordial no desenvolvimento das tecnologias atuais. Desenvolver uma API para um sistema abre grandes possibilidades de uso, dando à aplicação um nível mais razoável de acessibilidade e usabilidade. Dessa forma um mesmo serviço pode ser acessado por diferentes aplicativos e ter diferentes interfaces com os usuários. De maneira geral, uma API busca ajudar não apenas a pessoa que está programando a economizar

tempo, como também a empresa que irá criar aplicativos a reduzir custos, dando foco na qualidade do seu produto e na satisfação do cliente. Portanto, através desse projeto, busca-se desenvolver uma API que tem como objetivo ajudar a responder o seguinte problema: Como poupar tempo e estimular futuros desenvolvedores a criarem aplicativos capazes de se integrar a um sistema?

O SIGA (Sistema Integrado de Gestão Acadêmica) é o sistema utilizado pelos alunos e professores da FATEC (Faculdade de Tecnologia) para gestão do ambiente educacional como: o controle de presença; administração das notas; repositório de material de estudo e apoio; processo de rematrícula; etc. Atualmente, é utilizado por mais de 80 mil alunos e 3300 professores[1], espalhados em 66 campus. As informações presentes nesse sistema podem ser utilizadas para vários fins e, por esse motivo, esse projeto tem como objetivo a criação de uma API, para permitir que qualquer aplicação obtenha dados do SIGA, especificamente do perfil aluno. Serão disponibilizados, apenas, os métodos para obtenção de dados gerais (nome, curso, período, registro do aluno, percentual de conclusão do curso, percentual de rendimento, maior percentual de rendimento do curso), notas parciais e faltas parciais.

O primeiro capítulo desse projeto apresentará, como suporte teórico e embasamento técnico, as pesquisas realizadas em livros, revistas e materiais o*n-line*, fundamentais para o entendimento do objeto de estudo. Na parte inicial será possível verificar um breve histórico e as funcionalidades dos métodos e tecnologias empregados nesse projeto. Como mandatório, o segundo capítulo apresentará a metodologia de pesquisa utilizada para a construção da API. Na segunda seção do projeto busca-se demonstrar: a análise literária das peças utilizadas na pesquisa; o tipo de pesquisa realizada; o programa de desenvolvimento sugerido; custos projetados; testes e riscos. O terceiro capítulo exibirá as documentações e instruções técnicas fundamentais para a elaboração do mecanismo de conexão abordado nessa proposta. Por fim, o leitor encontrará as considerações finais obtidas, através das análises realizadas, após a implantação e teste da API.

1. Informação coletada no portal do Centro Paula Souza (2015).

1. FUNDAMENTAÇÃO TEÓRICA

As subseções abaixo apresentarão um resumo das informações coletadas das ferramentas, tecnologias e padrões que serão utilizados nesse projeto. Esse capítulo aborda os assuntos interligados ao tema escolhido para esclarecer e justificar o problema em estudo, pois “o uso de termos apropriados, de definições corretas, contribui para a melhor compreensão da realidade observada” (LAKATOS; MARCONI, 2007, p. 162).

1.1 Princípios para o profissional de TI

Segundo Comparato (2006), além dos conhecimentos técnicos, o profissional de TI (Tecnologia da Informação) deve apresentar também competências humanas, como: cordialidade, honestidade, respeito e dignidade. Analisar o comportamento e conduta ética dos profissionais que lidam com a tecnologia se tornou importantíssimo.

Fazem parte das obrigações deste tipo de profissional: assegurar a privacidade e proteção das informações, sejam elas organizacionais ou dos clientes; garantir a segurança destas informações; conhecer a legislação que regulamenta suas atividades profissionais e as normas da empresa em que trabalha; e assegurar o correto licenciamento das aplicações e equipamentos.

Mesmo não possuindo leis específicas, a área de tecnologia possui normas e regulamentações que precisam ser seguidas. Como princípios para esse projeto, a ética profissional, o acesso aos dados de terceiro e as informações para um correto licenciamento serão apresentados a seguir, pois esses aspectos podem não ser perceptíveis para o usuário final, mas são peça fundamental para estruturação de um projeto dentro dos requisitos mínimos esperados pelos utilizadores de um sistema.

1.1.1 Ética

Os profissionais brasileiros atuantes na área de TI ainda não dispõem de um código de ética que os discipline, ao contrário de outras categorias profissionais, como, por exemplo, a dos advogados. No entanto, tal situação não implica na total desregulamentação dos profissionais de tecnologia, na medida em que várias

empresas dessa área dispõem de códigos de ética autônomos e aplicáveis apenas aos seus próprios funcionários. Para Masiero (2000), essa formulação geralmente contempla seis aspectos básicos de obrigações éticas, que, em alguns momentos, conflitam entre si, deixando a priorização para o bom senso de cada profissional.

Essas seis obrigações estão divididas entre: a preocupação com o bem estar das pessoas em geral, quando consideradas usuários de sistemas computacionais (*hardware* e *software*); a proteção dos interesses do empregador em situações em que muitas vezes o empregador não tem habilidade para supervisionar tecnicamente o trabalho do profissional; os interesses do cliente, quando o profissional trabalha como consultor ou prestador de serviço autônomo; a própria organização e seus associados; o respeito aos colegas da mesma profissão e à colaboração entre colegas, que normalmente partilham os mesmos interesses; e o respeito à profissão em geral, tratando de aspectos do comportamento ético que devem ser evitados para não denegrir a profissão em si. (MASIERO, 2000)

Nesse projeto, a temática é a construção de meios de acesso e coleta de dados de um sistema educacional. Contudo, a estrutura utilizada não armazenará ou divulgará informações. Esses mecanismos servirão apenas como uma ponte entre a fonte de dados (SIGA) e a eventual aplicação desenvolvida.

Considerando o cenário citado acima, é necessário garantir que os dados trafegados entre uma ponta e outra seguirão as obrigações éticas aqui abordadas, seja na manipulação de dados de terceiro, bem como a garantia de sua proteção, com técnicas criptográficas e a não armazenagem de dados dos usuários. O acesso utilizado será somente de leitura, garantindo assim que o sistema da instituição de ensino em questão, a FATEC, não corra risco de invasões, ou de acessos não autorizados, respeitando assim a organização educacional.

1.1.2 Acesso a dados de terceiro

A proteção aos dados dos usuários faz parte de diversas normas, regulamentações e, se existentes, legislações de segurança voltadas ao ambiente de tecnologia. Essa importância se dá pelo próprio valor da informação e da garantia à privacidade resguardada por lei.

> Na sociedade da informação, ao mesmo tempo em que as informações são consideradas o principal patrimônio de uma organização, estão também sob constante risco, como nunca estiveram antes. Com isso, a segurança da informação tornou-se um ponto crucial para a sobrevivência das instituições. (TRIBUNAL DE CONTAS DA UNIÃO, 2007, p.70, apud WALTON, 2007, p.3)

Como a API criada nesse projeto utilizará as informações de autenticação do usuário, composta por número de identidade e senha, uma forma de acesso que não armazenará essas informações será utilizada. O detalhamento dessa tecnologia está presente no tópico de segurança.

Os dados coletados do usuário, após a autenticação, serão devolvidos à aplicação solicitante que deverá garantir a correta manipulação da informação e a adoção dos métodos de segurança necessários.

Diante do objetivo que, em resumo, será a construção de mecanismos de acesso e coleta de dados do sistema SIGA, as garantias de não divulgação ou armazenagem dos dados dos usuários ficarão sob a responsabilidade do programador.

O utilizador de uma aplicação, construída com os mecanismos aqui desenvolvidos, deverá verificar os termos de uso disponibilizados, visto que esse projeto não tem como objetivo a construção de um sistema, mas de métodos que poderão ser utilizados por outros programadores. Esses métodos não armazenam e nem divulgam informações sem que o programador, que os utilize, assim determine.

1.1.3 Licenciamento

Em TIC (Tecnologia da Informação e Comunicação), quase sempre é necessário a utilização de um determinado *software* para que uma tarefa ou trabalho seja executado, porém, dependendo da aplicação escolhida, pode ser que seja necessário efetuar um pagamento ao seu desenvolvedor para que ela possa ser utilizada. Isso é uma forma de recompensar o desenvolvedor pela criação do programa. O não pagamento faria com que o *software* estivesse sendo utilizado sem autorização, o que para Orrico (2004, p. 59) “é a prática ilícita, caracterizada pela reprodução ou uso indevido de programas de computadores, legalmente protegidos,

em outras palavras, é a reprodução ou utilização, não autorizada de software de outrem [...]".

Alguns *softwares* adotam licenças que não exigem pagamento para sua utilização, por esse motivo são conhecidos como *softwares* livres, definidos por Afonso et al. (2010) como qualquer programa computacional que permita a cópia, uso, estudo e redistribuição com apenas algumas restrições. Essa liberdade é o conceito central de *software* livre. O comum, na distribuição de programas desse tipo, é anexar à aplicação uma licença livre deixando o código fonte do programa disponível.

Ainda segundo Afonso et al. (2010), um *software* só é considerado livre quando atende quatro tipos de liberdade para seus utilizadores, sendo estas:

- Liberdade para executar o programa para qualquer propósito (liberdade nº 0).
- Liberdade para estudar o funcionamento do programa e adapta-lo para servir às necessidades individuais de cada utilizador (liberdade nº 1).
- Liberdade para redistribuição, inclusive venda de cópias de modo a ajudar o próximo (liberdade nº 2).
- Liberdade para modificar o programa e disponibilizar essas modificações, de modo a ajudar a comunidade (liberdade nº 3).

Entre as licenças livres mais utilizadas, a API estará sob a MIT\X11 que tem como principais características:

1.1.3.1 MIT/X11

Essa licença foi criada pelo MIT (Massachusetts Institute of technology). Segundo *Open Source Initiative* (2016, tradução própria) a MIT/X11 é uma licença permissiva que autoriza qualquer pessoa, que possua uma cópia do *software* e também sua documentação, utilizar um programa sem restrição alguma, desde que continue com um aviso de *copyright* e uma cópia da licença.

1.2 Linguagem de programação: Java

Como o produto desse projeto é um mecanismo lógico de programação, o desenvolvimento dessa proposta utilizou um conjunto de regras sintáticas e semânticas, por meio de algoritmos, chamado de linguagem de programação. Essa seção tem como objetivo discorrer sobre a linguagem utilizada, demonstrando o motivo da escolha. Essa parte do projeto foi dividida em duas etapas, a primeira parte contendo um breve histórico da linguagem escolhida e a segunda parte contém as características que ela possui.

1.2.1 História

O projeto para a criação do Java começou em 1991 na Sun Microsystems, chamado de Green Project. O intento era liderado por desenvolvedores que são considerados os mentores da linguagem, eram eles: Patrick Naughton, Mike Sheridan e James Gosling. Segundo Claro e Sobral (2008), era uma aposta da Sun na *web*, visto que todos seus projetos eram voltados para a WWW (World Wide Web), ou também chamada, apenas, de *web*. O nome inicial escolhido por James Gosling era *OAK*, que em inglês significa carvalho. A escolha teria sido por causa de uma árvore que James observava pela janela de seu escritório.

Porém, a nova linguagem só foi anunciada pela Sun em 1995. O nome Java vem de uma homenagem às xícaras de café que foram muito ingeridas por seus desenvolvedores durante todo o seu processo de criação. Logo, a linguagem se popularizou mais rápido que qualquer outra na história da computação. A sua popularidade foi, em grande parte, atribuída pelo seu uso na internet e, hoje, está presente na grande maioria de sistemas para computadores.

1.2.2 Características

Para Claro e Sobral (2008), quando uma nova linguagem surge, é muito comum que seus criadores tenham como base linguagens já conhecidas e que tiveram sucesso entre programadores. Com o C++ foi assim. Seu criador utilizou como base a linguagem C, realizando algumas melhorias. Isso também aconteceu com o criador

do Java, que fez uso de várias linguagens como base, sendo uma das principais o C++.

O Java é uma linguagem de simples e fácil manipulação, de acordo com Claro e Sobral (2008), pois contém uma sintaxe parecida com C++, porém, o Java é muitas vezes considerado uma versão mais simplificada do C++, por não conter os elementos, existentes em C++, que aumentam a dificuldade para os programadores. É orientada a objetos, o que permite que os desenvolvedores foquem nos dados a serem trabalhados, além de ter funcionalidades concedidas através de APIs que facilitam na manipulação de redes. Segundo Claro e Sobral (2008), essas APIs, de simples manipulação, facilitam o manejo de protocolos de redes.

Claro e Sobral (2008) explicam que a atração dos desenvolvedores para essa linguagem se dá pelo fato dela ser “multiplataforma”, ou seja, pode ser usada em qualquer sistema operacional. Além disso, a fama do Java cresceu rapidamente pelo fato da *web* estar em forte ascensão e o Java possibilitar fazer diversas coisas na *web* que, até um tempo atrás, não eram possíveis.

Em relação à segurança no Java, existem vários mecanismos que ajudam a tornar os aplicativos seguros. Um deles é a restrição da linguagem só ser compilada e interpretada através do JVM (Java Virtual Machine). O que garante que o código escrito não tenha acesso aos dispositivos de saída, entrada ou memória, deixando toda essa comunicação para o JVM, que decide o que pode ser feito ou não.

1.3 Segurança

A segurança é grande desafio no desenvolvimento de um sistema. É difícil construir um sistema 100% seguro e, em teoria, a única coisa que pode ser feita é tornar o ataque um trabalho difícil, segundo Siriwardena (2014, tradução própria).

Essa seção tratará sobre os mecanismos utilizados para tornar o produto desse projeto resistente à uma possível tentativa de interceptação dos dados de acesso do usuário, demonstrando a criptografia que será empregada e a chave a ser usada na respectiva criptografia.

1.3.1 Criptografia

O termo criptografia vem da junção das palavras gregas "kryptós" "gráphein" que, juntas, significam: oculto e escrever. Segundo Alecrim (2005) é um conjunto de conceitos e técnicas que visa codificar uma informação de forma que somente o emissor e o receptor possam acessá-la, evitando que um intruso consiga interpretá-la. A mensagem original fica ilegível se o processo de decodificação não for realizado, conforme exemplo da Figura 1.1. Para isso, uma série de técnicas são utilizadas e muitas outras surgem com o passar do tempo.

Figura 1.1 – Exemplo de um texto criptografado

Fonte: Alecrim, 2005, p.1

Na computação, a criptografia funciona a base de chaves criptográficas, onde um conjunto de bits, criados a partir de um determinado algoritmo, é capaz de codificar e decodificar informações. Para que a mensagem seja decodificada, uma chave compatível é necessária. De acordo com Alecrim (2005), com o uso de chave, um emissor pode usar o mesmo algoritmo (método) para vários receptores, necessitando apenas que cada um receba uma chave diferente. É possível manter o algoritmo, mesmo que um receptor perca sua chave, sendo necessário, apenas, a troca da chave.

Existem dois tipos de chaves criptográficas que, segundo Garrett (2012), devem ser de conhecimento das pessoas que utilizam internet e necessitam de proteção criptográfica, são elas as chaves simétricas (chave privada) e assimétricas (chave pública). Do ponto de vista de Cruz (2009) podemos dar a seguinte definição para essas chaves:

a) **Criptografia simétrica, ou de chave privada:** sistema que usa uma mesma chave para cifrar e decifrar a mensagem. Essa chave, portanto, deve ser de conhecimento tanto do emissor quanto do receptor.

b) **Criptografia assimétrica, ou de chave pública:** sistema baseado em algoritmos que habilitam o uso de duas chaves: uma pública, para cifrar a mensagem; e outra, diferente, mas matematicamente relacionada, a chave privada, para decifrá-la. Assim cada usuário possui um par de chaves: uma é abertamente divulgada, a chave pública; a outra é mantida em segredo, a chave privada. Esse sistema também pode ser usado para executar assinaturas digitais e troca de chaves entre correspondentes.

1.3.2 OAuth 2.0

Descrito por Jacobson, Brail e Woods (2011, tradução própria), poucas são as APIs que não requerem alguma forma de identificação, mesmo um simples registro. Porém, na maioria dos casos, uma, ou mais, dessas três técnicas básicas de segurança são utilizadas:

- Identificação: Quem está fazendo a solicitação?
- Autenticação: Você é quem diz ser?
- Autorização: Você pode fazer o que está tentando fazer?

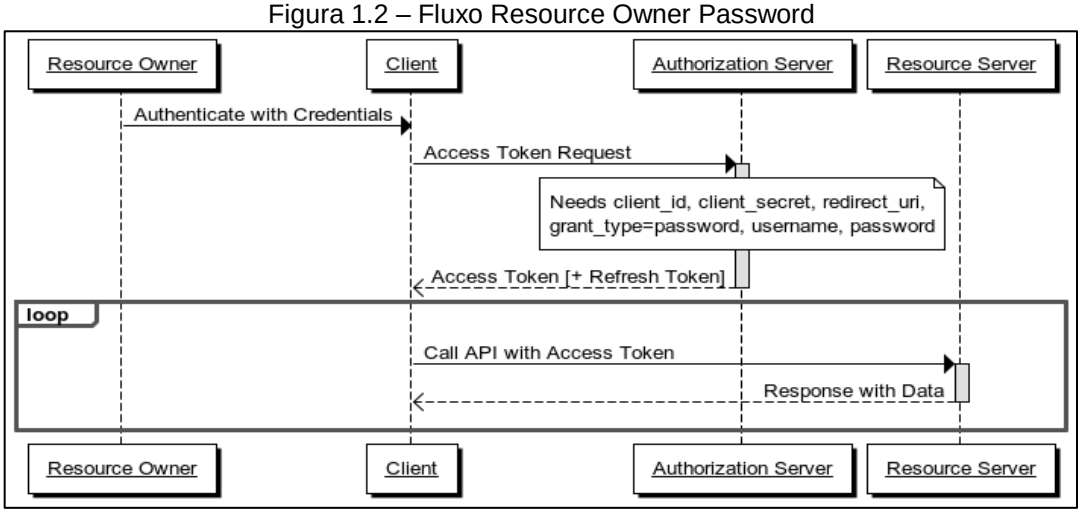
A autenticação vai determinar se quem está acessando a API é realmente quem diz ser. Existem vários métodos que podem ser utilizados para esta tarefa e, dentro destes, alguns incluem a aplicação do nome de usuário e senha do utilizador. Nesse projeto será utilizado o método chamado de OAuth 2.0. Utilizado em muitas das APIs atuais, como as do Facebook, é um protocolo aberto que permite, entre a API e aplicações *web*, uma autorização segura através de um mecanismo simples. Jacobson, Brail e Woods (2011, tradução própria) descreve como um aperto de mão entre a aplicação e o sistema, permitindo que a aplicação possa saber com que está se comunicando.

De maneira técnica, pode-se dizer que a autenticação é sempre feita através da transmissão da senha e nome do usuário entre aplicativo, API e sistema. Porém, com o OAuth 2.0, sempre que uma solicitação é feita do lado da aplicação destino (a que está solicitando os dados), ao invés de trafegar pela rede o nome do usuário e senha, um *token* de segurança é utilizado. Este *token* é obtido pela API que o utilizará

para representar o usuário no sistema fonte (o que possui as informações do usuário) (SPASOVSKI, 2013, tradução própria).

Dentro deste protocolo podem ser usados vários fluxos diferentes para a autenticação entre as partes. O fluxo que será abordado é chamado de Resource Owner Password Client, no qual, a senha não é armazenada na API, ficando apenas armazenada no aplicativo do usuário. A aplicação recebe da API um endereço, em formato HTTPS (Hyper Text Transfer Protocol Secure), do servidor de autorização, onde o *login* de usuário e senha do utilizador devem ser inseridos. Em seguida, se autorizado pelo serviço de autenticação, um *token* será devolvido para o aplicativo solicitante e para a API. O acesso ao sistema, para solicitar e receber as informações do usuário, estará, então, liberado.

Figura 1.2 – Fluxo Resource Owner Password



Fonte: Dophine, 2014

Na Figura 1.2 é possível ver como funciona o fluxo do Resource Owner Password. Um exemplo dessa tecnologia pode ser visualizado quando a opção “acessar com o Facebook” é utilizada. Este método direciona o usuário para uma página HTTPS onde as informações de entrada devem ser inseridas. Se as credenciais foram aceitas, o sistema, como o Facebook nesse exemplo, entrega um *token* de acesso à aplicação solicitante. O aplicativo poderá, então, fazer acesso ao servidor de dados do sistema sem ter acesso aos dados de *login*. Desta maneira, a API não será responsável por armazenar o usuário e senha do utilizador, tendo apenas contato como *token* de autenticação. Essa chave é provisória eliminando a necessidade de armazenar dados confidenciais do usuário.

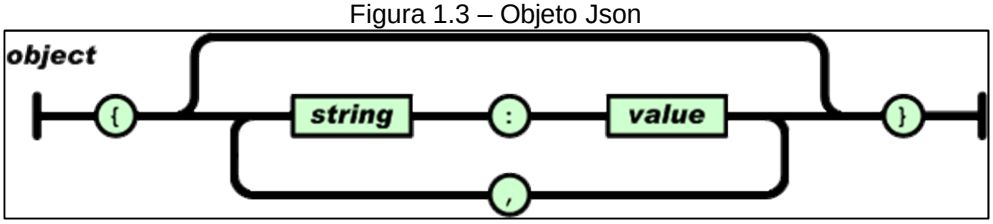
1.3.3 JWT (Json Web Token)

Json (JavaScript Object Notation, ou notação de objetos JavaScript, em português) é um formato de troca de dados bem leve, não exige muita memória, utilizado para guardar, enviar e receber informações. Fortemente adotado por aplicações *web* e APIs de empresas como Google, Facebook, Yahoo e Twitter. Apesar do nome (JavaScript), o formato pode ser usado em diversas linguagens de programação como C, C++, C#, Java e muitas outras. Conforme descrito por Miceli (2016), o JWT (JSON WEB TOKEN) é um formato popular e de fácil leitura pelos *websites*. Usado para codificar um *token* recebido pelo OAuth, sendo que, atualmente, grande parte das plataformas possuem um compilador Json, tornando essa utilização um benefício.

Nesse projeto, os *tokens,* gerados pelo mecanismo OAuth, apresentado no tópico anterior, utilizarão o método de criptografia assimétrica JWT. Segundo Miceli (2016), este é um fluxo que usa o OAuth para ganhar autorização no sistema necessitado de forma segura. “É um padrão aberto que define uma forma compacta e autocontida para transmitir, de forma segura, informações entre duas partes” (JSON WEB TOKENS, p.1, 2016, tradução própria).

> Json é uma alternativa muito utilizada como contrapartida ao XML na troca de mensagens entre o cliente e a API. Um Objeto JSON é definido como um conjunto de nomes e valores separados por dois pontos e delimitados por vírgulas [...]”. (SAMPAIO, 2007, p. 115).

Figura 1.3 – Objeto Json



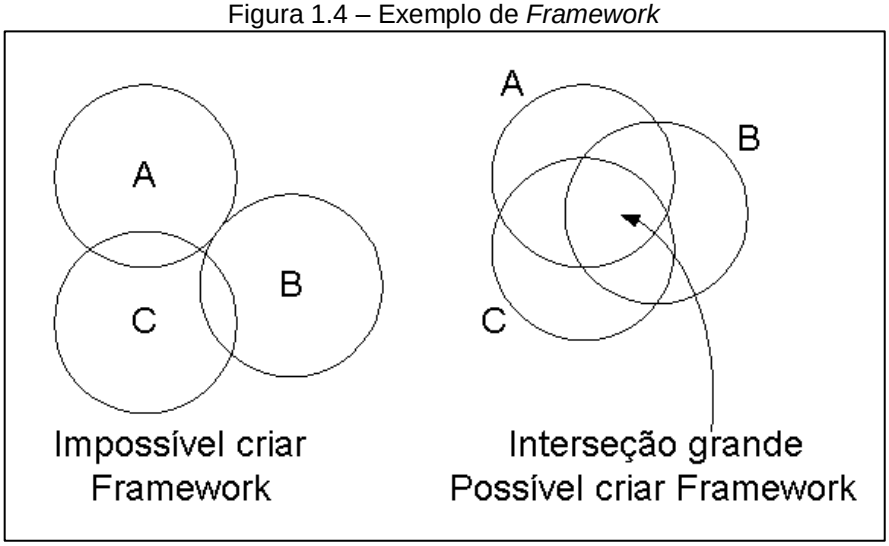
Fonte: ECMA INTERNATIONAL, 2013

A Figura 1.3 representa a estrutura de um objeto Json. Uma chave ({) determina o início do encapsulamento, seguida por uma variável (*string*) que deve ser separada, por dois pontos (:), do seu respectivo valor. Se existirem mais variáveis, estas, deverão ser separadas por vírgula (,). O resultado terá, como exemplo, a seguinte saída: {usuário:beltrano,senha:1234}.

1.4 *Frameworks* e Bibliotecas

Johnson e Foote (1988, tradução própria) descrevem *framework* como um conjunto de classes, com um formato abstraído, que busca solucionar problemas genéricos. De forma resumida, um *framework* é um conjunto de bibliotecas com uma base que, com poucos ajustes, pode acabar se tornando a solução para determinado assunto. Por exemplo, uma tela de cadastros. Esse tipo de tela sempre tem campos genéricos como nome, idade e sexo. Um *framework,* propõe um modelo para estes dados de forma que o desenvolvedor não se preocupe em criar tais formulários. O programador se preocupa, apenas, em acrescentar outros tipos de dados necessários para realizar seu objetivo.

Figura 1.4 – Exemplo de *Framework*

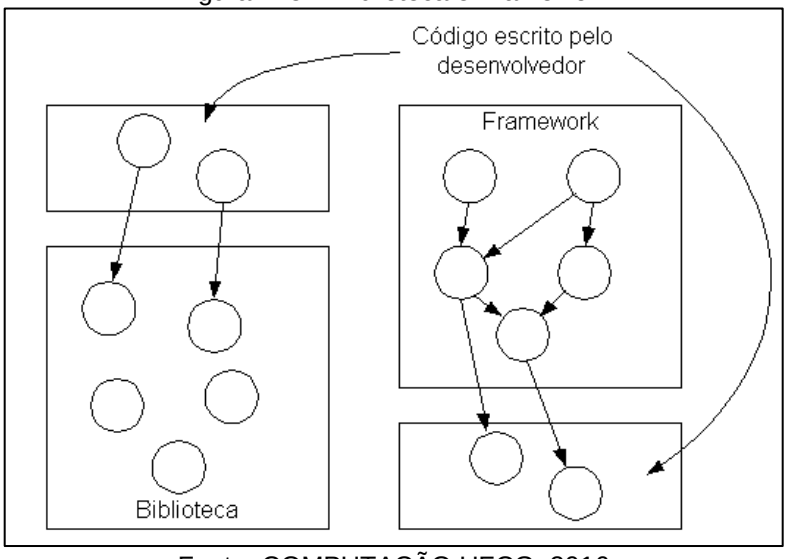


Fonte: COMPUTAÇÃO UFCG, 2016

A Figura 1.4 demonstra a possibilidade de criação de um *framework* quando *softwares* compartilham funções comuns. Com isso, o desenvolvedor necessita de alguns ajustes para tornar o *software* mais completo. Se não existir partilha de funções, não existe interseção, impossibilitando a criação de um *framework*.

Para Johnson e Foote (1988, tradução própria), uma biblioteca também pode ser classificada como um conjunto de classes, mas, diferentemente do *framework*, não necessita de correlações entre os itens ali presentes. Nela, é o desenvolvedor quem chama as classes para exigir uma determinada função, enquanto que no *framework,* o resultado da colaboração entre as classes é o que é resgatado pelo desenvolvedor.

Figura: 1.5 – Biblioteca e Framework



Fonte: COMPUTAÇÃO UFCG, 2016

Na Figura 1.5 é possível observar como um código escrito pelo desenvolvedor interage com as classes contidas em uma biblioteca. Também podemos verificar como as classes, contidas em um *framework*, interagem entre si, fornecendo determinada função ao código escrito pelo desenvolvedor.

1.4.1 *Framework* principal: *Spring Framework*

O *Spring Framework*, segundo Bachiega (2008), é um modelo de aplicação em camadas Java e, por ser completo, é uma das alternativas mais populares para um servidor de aplicações. O *Spring* tem código aberto e uso livre, uma vantagem que ajudou em sua popularização.

É uma ferramenta para auxiliar desenvolvedores, pois facilita a construção de aplicações. Com o uso efetivo de orientação a objetos de maneira leve, não intrusivo e configurável, o *Spring* é utilizável tanto em aplicações *web* quanto em desktops. É um *framework* que facilita a criação, pois tem um baixo número de acoplamentos entre os componentes (BACHIEGA, 2008).

1.4.2 Extração de dados de HTML: jsoup

O *jsoup* é uma biblioteca Java de código aberto. Escrita por Jonathan Hedley, o *jsoup* foi desenhado para extrair e manipular dados HTML. É uma ferramenta de extração de dados tão poderosa que é usada em inúmeros projetos da Google. Na

maioria das bibliotecas somente é possível capturar o conteúdo completo, impossibilitando a captura de um conteúdo específico, entretanto, com o *jsoup*, é possível capturar um conteúdo preciso e, também, o valor de seus atributos. (BALLEM, 2012).

Muitas páginas da internet e serviços virtuais, atualmente, fornecem informações através de RSS (Really Simple Syndication), ou através de APIs, porém, ainda existem alguns sites que não oferecem essas facilidades. Para casos onde se tem a necessidade de ferramentas para extrair os dados de um site, o Jsoup é um dos mais poderosos aparatos no quesito de extração e de transformação do conteúdo HTML (HOUSTON, 2013, tradução própria).

1.4.3 Automação de navegação *web*: Selenium

Uma das grandes ferramentas para automação de navegação é o Selenium. Ele normalmente é usado para automatizar testes, pois tem a capacidade para ser usado em múltiplos navegadores. É um instrumento capaz de automatizar qualquer interação que um programa possa ter com os mais populares navegadores.

Esta ferramenta possui uma função chamada de WebElements, utilizada para auxiliar a automação em elementos HTML. Este driver do Selenium permite interações em elementos *web* muito bem organizadas, como: localizar um elemento; coletar seus atributos; declarar textos em páginas virtuais; entre outros (SAMS, 2015).

Outra funcionalidade do Selenium é de automatizar interações, entre o programa e o navegador, sem a necessidade de um *software* de navegação, ou seja, simular uma navegação, em páginas de internet, sem um *browser*. Isso é possível, pois a ferramenta, através do *driver* Jbrowser, pode fazer essa navegação simulada.

1.5 Padrões e técnicas de desenvolvimento

É necessária, para o desenvolvimento de um *software*, a utilização de algumas técnicas de desenvolvimento. Nesta seção, dividida em cinco partes, serão abordadas as técnicas utilizadas para elaboração desse projeto. Essas técnicas são: Web Scraping, Cache, Logging, Inversão de Controle e Injeção de Dependência.

1.5.1 Web Scraping

Segundo Heaton (2002, tradução própria), uma grande quantidade de informação está facilmente acessível na internet atualmente. Elas são produzidas e consumidas por usuários humanos, através de um navegador.

No entanto, esses dados poderiam ser extraídos e utilizados para vários outros propósitos. Por exemplo, no site do vestibular da FATEC é possível consultar a demanda dos cursos por semestre e por unidade, mas apenas através do navegador e realizando várias vezes o mesmo processo. Através do Web Scraping, que consiste na extração (manual ou automática) de dados de um site para salvá-lo em outro lugar, com a ajuda de um programa. Segundo Webster (2005, tradução própria), seria possível, por exemplo, salvar todos os dados em uma planilha e realizar uma análise posterior de maneira mais simplificada.

De acordo com Mitchell (2015, tradução própria), a obtenção automática de dados da internet é tão antiga quanto à internet em si. Essa técnica já foi chamada de Screen Scraping, Data Mining, Web Harvesting, sendo que os esses termos, ou outras variações similares, também são citados por Webster (2015, tradução própria). Dessa forma, apesar de não ser um termo novo, Web Scraping, atualmente, é o termo de uso consensual, e os programas que utilizam essa técnica podem ser chamados de *bots* (MITCHELL, 2015, tradução própria).

> Para aqueles que não possuem a habilidade, programação de computadores pode parecer como um tipo de magia. Se programação é magica, então Web Scraping é feitiçaria, ou seja: a aplicação da magia para feitos particularmente impressionantes e úteis, ainda que, surpreendentemente, sem esforço" (MITCHELL, 2015, p. 8, tradução própria).

Mitchell (2015, tradução própria) conceitua que, na teoria, Web Scraping é a obtenção automatizada de dados de *websites* através de programas que realizam consultas, requisições, tratamento e extração dos dados necessários a partir do HTML obtido. Sua implementação poderia ser descrita, de maneira sucinta, em três passos: 1º. Obtenção do conteúdo HTML; 2º. Tratamento e extração de dados; 3º. Armazenamento dos dados obtidos;

Por fim, Web Scraping se torna necessário, pois nem sempre uma API estará disponível. Isso pode ocorrer por qualquer um dos motivos expostos por Mitchell (2015, tradução própria): conjunto pequeno de dados ou falta de capacidade técnica. É possível acrescentar a essa lista de motivos, a inexistência de demanda por uma API, ou mesmo a resistência humana à mudança de um sistema que já funciona.

1.5.2 Cache

A obtenção e processamento de dados pode ser algo computacionalmente custoso. Uma técnica bastante utilizada é o cache: fazendo com que, após obtidos ou processados, os dados sejam armazenados em um cache da aplicação. Esse, por sua vez, tem o funcionamento semelhante a uma memória cache.

Castro e Matos (2007, tradução própria) definem memória cache como sendo uma memória temporária, geralmente automática e oculta ao usuário. Fornece acesso rápido aos dados usados mais frequentemente e seu fluxo o é simples: primeiro os dados são procurados na memória cache e, caso não disponíveis, serão procurados na memória principal. A memória cache pode acelerar o acesso a qualquer tipo de dispositivo, até mesmo outro cache, segundo Jacob, Ng e Wang (2008, tradução própria).

A principal característica relacionado a um cache é a sua coerência. Isso significa, de acordo com Castro e Matos (2007, tradução própria), que qualquer leitura nele realizada deve retornar o valor da escrita mais recente. Dessa forma fica garantido que os dados obtidos sempre serão os mais atualizados possíveis.

1.5.3 Logging

Gupta (2005, tradução própria) diz que *logging* é uma forma sistemática e controlada de representação, de maneira legível, do estado de uma aplicação. Assim como um prontuário médico auxilia no acompanhamento do estado de saúde de um paciente, um registro das atividades de um *software*, principalmente de suas falhas, apoia os desenvolvedores em seu trabalho. Ainda segundo Gupta (2005), como um mecanismo para se verificar o funcionamento da aplicação de maneira geral, é importante o registro de *logs*, visto que nenhum *software* é livre de *bugs*

De acordo com Chuvakin et al. (2013, tradução própria), esses registros podem possuir as seguintes informações:

- O que aconteceu?
- Quando aconteceu?
- Onde aconteceu?
- Quem estava envolvido?
- De onde veio?

Segundo Gupta (2005, tradução própria), as principais vantagens obtidas através do registro de *logs*, são:

- Diagnóstico de Problemas
- Depuração rápida
- Histórico
- Fácil manutenção
- Diminuição de custo e prazo

1.5.4 Inversão de Controle

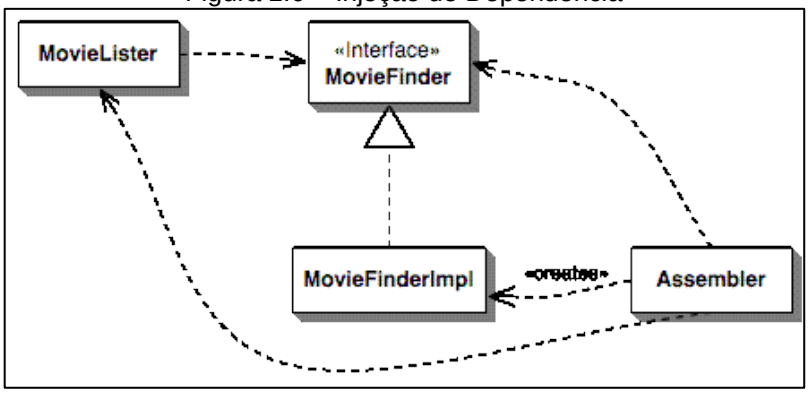
Inversão de controle, ou abreviadamente IoC (Inversion of Control), é um tipo de desenvolvimento muito utilizado em *frameworks*. Como o próprio nome sugere, esse modelo de desenvolvimento busca inverter o controle que determinado método do programa tem sobre uma função específica para outro método exercer. Torna cada um desses especializados em uma única função, tirando o controle deles sobre as tarefas que podem, ou não, executar. De acordo com Gama (2016), inversão de controle é, de forma abstraída, a mudança de conhecimento que uma classe tem sobre outra.

Para Fowler (2005, tradução própria) uma grande característica de Inversão de controle é a delegação, por parte do desenvolvedor, de uma tarefa para uma função de um *framework*, ou *software* de desenvolvimento. O modelo escolhido, então, é quem decide quando chamar e utilizar os métodos criados pelo desenvolvedor.

1.5.5 Injeção de Dependência

A Injeção de Dependência (Dependency Injection) é uma técnica de desenvolvimento para os casos onde os módulos do sistema necessitam que seja mantido um baixo nível de acoplamento. Nesta solução, as dependências dos módulos são definias pela configuração de uma infraestrutura de *software* que é responsável por injetar cada componente em suas dependências declaradas (NENE, 2005, tradução própria).

Figura 1.6 – Injeção de Dependência



Fonte: Fowler, 2004

A ideia básica, segundo Fowler (2004, tradução própria), de Injeção de Dependência como figura 1.6 é ter um objeto separado, um ajuntador (*assemble*), que preenche o campo na classe de listagem com a apropriada implementação para achar a interface localizadora, resultando em um diagrama de dependência ao longo das linhas.

1.6 Ferramentas

Nessa seção, dividida em quatro partes, as ferramentas utilizadas para o desenvolvimento serão apresentadas. A primeira parte abordará o IDE a ser utilizado, a segunda parte o Build, que irá ajudar na criação da estrutura, passando pelo controle de versão e terminando com o Servlet Container utilizado.

1.6.1 IDE: Eclipse

O IDE (Integrated Development Environment), que pode ser traduzido livremente como Ambiente de Desenvolvimento Integrado, de acordo com Novaes

(2014), é uma categoria de *software* criada com a finalidade de facilitar a vida dos programadores. São aplicações que possuem todas as funções requeridas para desenvolver programas de computadores e possuem recursos que ajudam a diminuir a ocorrências de erros nas linhas dos códigos.

Esse projeto utilizará o Eclipse, um IDE da IBM iniciado em 2001. A sua tecnologia foi baseada em *plug-ins* para desenvolvimento em Java, porém, como é código aberto, este IDE pode ser usado para outras linguagens com facilidade, sendo necessário, apenas, instalar *plug-ins* adicionais, fornecendo assim a variadas opções de uso.

O Eclipse será usado pelos seguintes motivos: ser um *software* livre; oferecer com facilidade o suporte para variados tipos de projetos, apenas adicionando os *plug-ins* necessários; apresentar uma interface de fácil manuseio e em constante evolução, sempre recebendo novos *plug-ins*.

De acordo com Faria et al. (2010), uma das desvantagens é o fato de não existir uma versão *online*, sendo necessário instalar um *software*, que não é leve, ou seja, requer muito espaço no disco rígido – ainda mais se for necessário a instalação de muitos *plug-ins* adicionais. Se mal instalados, os *plug-ins* também podem causar problemas na implementação.

1.6.2 Build: Maven

O processo de criação de programas em Java, conforme descrito por Ottero (2016), envolve: a criação de uma estrutura com diretório principal e vários subdiretórios; a configuração de vários arquivos XML (Extensible Markup Language); a obtenção de bibliotecas; a criação de pacotes e várias outras tarefas, que podem tornar a execução de um projeto um pesadelo. Essa técnica pode ser chamada de Build.

Para ajudar a solucionar os erros que podem ocorrer em todo processo de criação de novas aplicações, foram criadas ferramentas para ajudar na organização dos projetos. Vários mecanismos foram criados a partir do ano 2000 e, entre eles, um

dos primeiros conhecidos foi o Ant. Porém, ainda havia necessidade, e espaço, para muitas atualizações, culminado com o lançamento do Maven em 2004.

O Apache Maven, na visão de Ottero (2016), é uma ferramenta de automação e gerenciamento de projetos (*build*) Java e outras tecnologias. Fornece a seus desenvolvedores uma forma de automatizar, padronizar, construir e publicar suas aplicações. Também pode fornecer diversas funcionalidades através de *plug-ins* que estimulando o uso das boas práticas na organização.

O Maven auxilia na adoção de boas práticas no projeto, pois, segundo Ottero (2016), utiliza um conceito chamado programação por convenção. Esse conceito procura, com um modelo de desenvolvimento de aplicações, diminuir o número de decisões que os desenvolvedores precisam tomar. Um exemplo de uso seria a adoção de uma convenção de nomes. Os nomes das tabelas do banco de dados serão o plural de sua respectiva classe. Por exemplo, uma classe venda teria sua tabela chamada, por padrão, de vendas. Isso reduz decisões e ações necessárias do programador.

1.6.3 Controle de Versão: Git

O SVC (Sistema de Controle de Versão) tem como responsabilidade o gerenciamento e controle das versões na implementação de documentos. Para Pernambuco, Santos e Valente (2016), é uma prática comum na engenharia de *software*, pois ajuda a manter o código fonte atualizado para toda equipe de desenvolvimento esses sistemas. Essas ferramentas apoiam os desenvolvedores no controle das modificações realizadas nos arquivos do projeto e possuem mecanismos automatizados que garante a integridade e rastreabilidade dessas modificações ao longo do tempo

O Git é um SVC distribuído que foi projetado e desenvolvido por Linus Torvalds. Ele é distribuído como um *software* livre, sob os termos da GNU (General Public License). Um SVC distribuidor, ou descentralizado, como o GIT funciona, segundo Freitas (2010), com vários repositórios independentes e autônomos, um para cada usuário, e cada um desses repositórios tem uma área de trabalhado acoplada a

ele. Algumas das vantagens e desvantagens desse sistema de controle estão no Quadro 1.1.

Quadro 1.1 – Vantangens e Desvantagens GIT

Ponto de Vista	Vantagens	Desvantagens
Desenvolvedor	Rapidez	Necessidade de maior conhecimento da ferramenta e do processo
	Autonomia	
	Ramos individuais	
	Facilidade de mesclagem	
Coordenação/Gerência	Redução de custos com servidor e infraestrutura externa de rede	Necessidade de maior capacitação de desenvolvedores
	Confiabilidade	Importante ter um processo definido
	Aumento da produtividade	O controle de mudança ainda precisa ser centralizado

Fonte: DIAS (2016)

O Git é ideal para utilização no projeto, pois, conforme Toffolo (2011), tem em suas vantagens a velocidade. Ele é muito mais rápido que os outros *softwares*. Os seus algoritmos de compressão são eficientes, simples e possui o código aberto.

1.6.4 Servlet Container: Tomcat

O usuário apenas consegue solicitar páginas estáticas do servidor, o que não é suficiente se o usuário precisar de uma página baseada em alguma informação inserida. Para Program Creek (2013, tradução própria), a ideia básica do Servlet Containter é usar o Java para gerar, dinamicamente, uma página *web* no servidor. Então, o Servlet Container é, basicamente, um servidor *web* que interage com os *servlets*.

> Um servlet é uma classe escrita em Java cujos objetos têm a finalidade de gerar documentos codificados em HTML. Esta característica dos servlets implica que um *web designer* precisa conhecer Java para poder construir as páginas de uma aplicação. (Devmedia, 2016, p.1).

O Apache Tomcat é um *container web* de código aberto baseado em Java. Foi criado para executar aplicações virtuais que utilizam tecnologias *servlet* e JSPs (Java Server Pages). É um servidor bastante estável com todas as características que um *container* comercial possui (MEDEIROS, 2016).

Do ponto de vista operacional, a principal finalidade das tecnologias de *servlets* e JSP é permitir a criação dinâmica de conteúdo. A dinâmica, em um cenário típico, funciona do seguinte modo:

- Um usuário, no seu *browser*, solicita algum documento (indicado por um URL - Uniform Resource Locator) a um servidor Tomcat;
- O servidor, ao receber uma solicitação (URL) do usuário, executa o *servlet*, ou JSP, correspondente àquele URL (a associação entre URL e *servlet* ou JSP é especificada no arquivo web.xml).
- O conteúdo gerado pelo *servlet* ou JSP, normalmente um documento no formato HTML, é uma combinação de *tags* HTML (incluídos explicitamente) e o resultado de algum processamento (por exemplo, algoritmo Java e/ou acesso a um banco de dados).
- O usuário recebe o conteúdo gerado pelo servidor Tomcat e o exibe através do seu *browser*.

1.7 API

O objetivo dessa seção é ajudar explicar o conceito tema desse projeto. Clarificar o histórico e as propriedades dessa tecnologia que será peça chave no desenvolvimento de futuros aplicações.

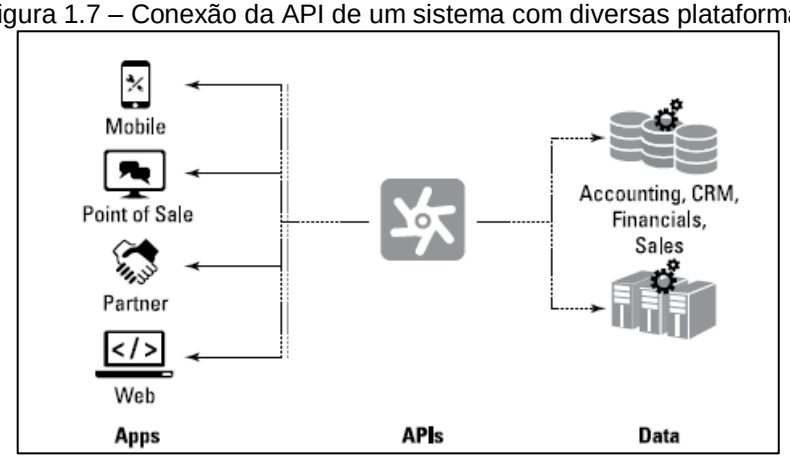
Se você já utilizou um aplicativo da Netflix, ou do Facebook, ou Twitter em seu celular, você já utilizou uma API. O significado Application Programming Interface, em inglês, pode ser traduzido como Interface de Programação de Aplicação. Em termos simples, pode ser definido como um carteiro que receba as cartas do posto de correio e as entrega de porta em porta.

A definição técnica para uma API, de acordo com Jacobson, Brail e Woods (2011, tradução própria), é um caminho para dois computadores conversarem entre

si, em uma rede (geralmente internet), usando uma linguagem comum na qual ambos lados entendam. Ainda segundo Jacobson, Brail e Woods (2011, tradução própria), uma API pode prover para desenvolvedores internos, parceiros ou contratados acessos à informações e serviços para desenvolvimento de novas aplicações com muita mais velocidade.

Nijim e Pagano (2014, tradução própria) descrevem API como uma maneira segura, confiável e eficiente de conectar serviços, processos, conteúdo e informações com seus canais de parceiros, times internos e desenvolvedores independentes, conforme demonstrado na Figura 1.7. Este fator é o que faz grandes empresas tornarem as APIs um fator comum na chave para compartilhar informações e construir consistentes canais de experiências com clientes.

Figura 1.7 – Conexão da API de um sistema com diversas plataformas



Fonte: Nijim e Pagano, 2014, p.10

A descrição dada por Nijim e Pagano (2014, p.10, tradução própria), sobre o API no mercado, é: “Pense na API como uma cola digital na cadeia de valor digital, energizando aplicativos de telefones e *web*, conectando sistemas e permitindo inovação”.

As APIs, ainda segundo os autores, são fundamentais para:

- Integração de conteúdo entre parceiros para criação de vendas casadas e oportunidades de vendas;
- Criar novas linhas de negócios e estendendo ofertas de produtos, através do aproveitamento de informações corporativas;

- Fortalecer a marca provendo uma consistente, familiar e personalizada experiência através aplicativos;
- Permitindo a reutilização, resultando em novas integrações parceiras em dias, não em semanas.

Uma API deve ser tratada como um *software*, tendo em conta: versão; compatibilidade; tempo para acomodar novas funcionalidades; e um balanceamento entre o suporte da base existente e nas mudanças necessários para evoluções (JACOBSON; BRAIL; WOODS, 2011, tradução própria).

Como um *software*, Jacobson, Brail e Woods (2011, tradução própria) define que a API precisa seguir algumas especificações, são elas:

- A funcionalidade que a API vai oferecer;
- Descrever quando a funcionalidade vai estar disponível e quando pode mudar para modo incompatível;
- Descrever restrições técnicas que API possa ter;
- Descrever restrições legais ou de negócios, como limitações de marcas, tipos de uso, etc;
- O provedor da API deve também fornecer outras ferramentas como:
- Mecanismos para acessar e entender os termos de uso da API;
- Documentação para ajudar na compreensão da API;
- Informação operacional sobre a saúde e quanto usada a API está sendo;
- A estrutura da API é parte de um contrato. Este contrato é obrigatório, e não pode ser alterado casualmente.

1.7.1 Pública x Privada

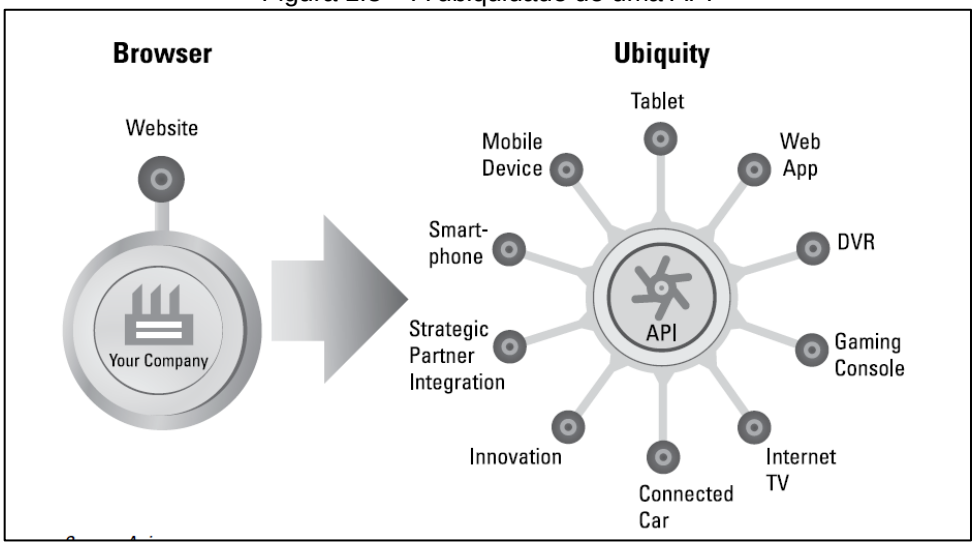
As APIs podem ser publicadas em dois tipos: privada e pública. Sendo as privadas as que prevalecem no mercado, segundo Jacobson, Brail e Woods (2011, tradução própria). Grandes empresas usam muito mais as do tipo privada, para ajudarem seus negócios, do que as públicas, mesmo a última sendo visível ao grande público. As APIs públicas são apenas a ponta do iceberg, enquanto as privadas são a grande massa encoberta pela agua.

Quando se diz pública, significa que a API está disponível para quase todos, com pouco ou com nenhum acordo contratual (além do acordo de termos de uso) com o provedor da aplicação. A API Privada é usada de várias maneiras, seja para suportar esforços internos, ou de parceiros, no uso da API. Para os parceiros é necessário um contrato legal. Por fim, o privado e público se referem a formalidade de contratos legais, não se referindo ao uso do conteúdo, ou dos aplicativos, desenvolvidos para usá-la (JACOBSON; BRAIL; WOODS, 2011, tradução própria).

1.7.2 API x *Website*

Existe muitas diferenças entre um *website* e uma API. Segundo Jacobson, Brail e Woods. (2011, tradução própria). O *website* oferece conteúdo sobre demanda. No *website* não existe contratos ou estruturas que usam seu conteúdo. Se o site mudar, o visitante que chegar vai receber o novo conteúdo, e seus navegadores não serão afetados. Nenhuma mudança é visível ao usuário. Com a API é muito diferente porque existe um contrato, e os programas devem ser construídos em cima desse contrato. Se você alterar o contrato dessa API, o efeito em cimas desses aplicativos é potencialmente grande.

Figura 1.8 – A ubiquidade de uma API



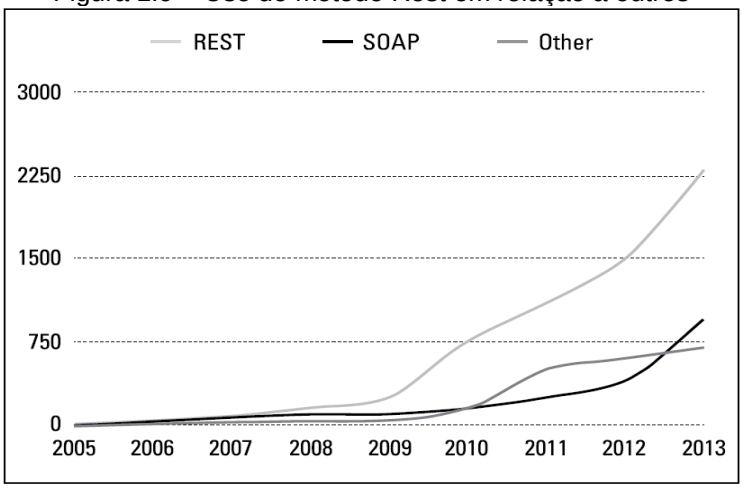
Fonte: Nijim e Pagano, 2014, p.17

A Figura 1.8 ilustra, para Nijim e Pagano (2014, tradução própria), as oportunidades presentes na transição, de um simples *website*, para um modelo corporativo totalmente conectado que a API pode lhe trazer.

1.7.3 Arquitetura REST

As APIs podem ser escritas usando duas tecnicas: a *Representational State Transfer* (REST), em português Transferência de Estado Representacional; e *Simple Object Access Protocol* (SOAP), traduzido como Protocolo Simples de Acesso a Objetos. Para Nijim e Pagano (2014, tradução própria), o protocolo REST é o mais utilizado conforme ilustrado na Figura 1.9.

Figura 1.9 – Uso do método Rest em relação a outros



Fonte: Nijim e Pagano, 2014, p.38

O estilo REST foi desenvolvido pelo PhD Roy Fielding, que foi também um dos autores do protocolo HTTP. O que, consequentemente, fez com que o REST fosse baseado nos padrões HTTP, que é um dos protocolos mais usados para comunicação entre computadores. Hoje, qualquer plataforma computacional consegue se comunicar com um servidor HTTP (JACOBSON; BRAIL; WOODS, 2011, tradução própria).

Uma das principais fontes para o design de uma API, para Mulloy (2012), é o REST. Porque é um estilo de arquitetura, não ficando restrita a padrões, que permite muita flexibilidade. Por causa dessa flexibilidade e liberdade de estrutura, existe um grande apetite para melhores práticas de design.

2. METODOLOGIA

As metodologias se utilizam de conhecimentos adquiridos no decorrer de séculos e aperfeiçoam-se por meio de contribuições, cada vez mais constantes, de acadêmicos, doutores, universidades e cientistas. Tem como função o auxílio na realização e exposição de pesquisas, com um foco voltado para a eficiência dos resultados. Nesse capítulo do projeto será demonstrada a metodologia utilizada para a condução da pesquisa, pois esse item, segundo Lakatos e Marconi (2007, p.221), "[..] responde às seguintes questões: Como? Com quê? Onde? Quanto? ".

2.1 Análise literária

Realizada durante o desenvolvimento do projeto de conclusão, a fundamentação literária foi necessária para embasamento teórico e ferramental. Por ser um tema pouco explorado no Brasil e, principalmente, no meio acadêmico, a principal fonte de informações foram literaturas estrangeiras e sites especializados.

2.2 Tipo de pesquisa

Segundo Gil (2002, p.41), "Com relação às pesquisas, é usual a classificação com base em seus objetivos gerais. Assim, é possível classificar as pesquisas em três grandes grupos: exploratórias, descritivas e explicativas". Adotamos a o método exploratório, pois, segundo Cervo, Bervian e Silva (2007, p.61),

> Esta pesquisa não requer a formulação de hipóteses para serem testadas, ela se restringe por definir objetivos e buscar mais informações sobre determinado assunto de estudo, portanto ela seria um passo inicial para o projeto de pesquisa. A pesquisa exploratória é recomendada quando há pouco conhecimento sobre o problema a ser estudado.

A pesquisa documental assemelha-se muito à pesquisa bibliográfica. A maior diferença entre elas está na categoria das fontes. A pesquisa bibliográfica se utiliza fundamentalmente das contribuições dos autores sobre um determinado assunto, já a pesquisa documental utiliza matérias que ainda não foram avaliados, ou que ainda podem ser reelaborados de acordo com os objetos da pesquisa (GIL, 2002).

Foram utilizados os dois métodos de pesquisa tanto o bibliográfico quanto o documental. Isso porque existem pouquíssimas fontes que tratem do tema estudado.

Foi necessário buscar recursos bibliográficos estrangeiros, em consequência da falta de informações técnicas em português.

Como apoio, foram utilizadas, também, outras teses, artigos acadêmicos e fóruns técnicos da área de TI, visando um melhor desenvolvimento e absorção de conteúdo. Os dados obtidos para o progresso documental foram exclusivamente de ordem secundária, o projeto não possuí fontes quantitativas e qualitativas devido à complexidade, no que tange a ferramenta e seu uso.

2.3 Programa de desenvolvimento

Como o objetivo desse projeto é a criação de um conjunto de métodos que permita a obtenção de dados do sistema SIGA, através de uma API Rest, a divisão das fases de desenvolvimento se baseará nos seguintes módulos:

- Fase 1: Elaboração do conector SIGA. Módulo responsável por realizar o acesso e obtenção dos dados do SIGA através de Web Scraping. Será desenvolvido em Java e utilizará os mecanismos: Selenium (para simulação de navegação) e Jsoup (para extração dos dados dos HTMLs obtidos durante a navegação simulada). O conector será responsável por todo acesso ao SIGA, dessa forma outros módulos somente farão acesso através desse.
- Fase 2: A criação do servidor de autenticação. Sistema *web*, desenvolvido em Java com o *framework Spring*, será responsável pelo processo de autenticação e autorização de acesso à API, através do protocolo OAuth2. Nele o usuário digitará suas credenciais de acesso ao SIGA e, caso o acesso seja bem-sucedido, será questionado se o sistema/aplicativo utilizado pode ter acesso a esses dados.
- Fase 3: Definição da API. Documentação escrita na ferramenta Swagger e disponibilizada nos formatos HTML e PDF. Irá conter a descrição completa de todos os métodos presentes na API e como ela deverá ser utilizada.
- Fase 4: Servidor API. Sistema *web*, desenvolvido em Java com o *framework Spring*, será responsável por implantar os métodos definidos para a API, seguindo a arquitetura REST. O acesso será restrito somente a requisições contendo informações válidas de autenticação.

2.4 Custos

Segundo Femenick (2005), a utilização de meios financeiros na aquisição de materiais, trabalhos e serviços, entram como custos de um projeto.

Por se tratar de um projeto de cunho acadêmico e acesso restrito aos alunos da Faculdade de Tecnologia Adib Moises Dib, a hospedagem dos serviços *web* pode, atualmente, ser feita com o custo de R$ 9,90 (nove reais e noventa centavos) ao mês, no plano básico do UOL Host (empresa de hospedagem, serviços de criação de sites, computação na nuvem e infraestrutura de comércio eletrônico). Além disso, é necessário a aquisição de um domínio para o sistema, ao custo de R$ 40,00 (quarenta reais) por ano, a partir de 1º de janeiro de 2017, pagos ao Registro.br (entidade responsável por registrar e efetuar a manutenção de domínios brasileiros assim como distribuem endereços IP).

A aquisição de um plano de hospedagem anual, junto ao UOL Host, custará R$ 118,80 (cento e oitenta reais e oitenta centavos), gerando como bônus a contratação do primeiro ano de registro de domínio. Dessa forma, no primeiro ano, será necessário o valor da hospedagem, na quantia de R$ 118,80 (cento e dezoito reais e oitenta centavos) e, nos anos seguintes, será somado a esse montante, o custo equivalente ao registro do domínio, totalizando R$ 158,80 (cento e cinquenta e oito reais e oitenta centavos).

Os valores mencionados são válidos a partir de 02/11/2016, e não leva em consideração possíveis reajustes relacionados à mudança na política de preços das empresas fornecedoras.

2.5 Testes

Para Pressman (2011), a qualidade de um produto refere-se a: características especificadas para a construção de um artefato, a qualidade do material, e as tolerâncias aceitas e no desempenho esperado. Todos esses fatores vão influenciar na qualidade do produto.

Ainda segundo o autor, a qualidade do projeto faz parte do grau de atendimento das funções e características especificadas no modelo de requisito. Já a qualidade de conformidade está focada no grau em que a implementação segue o projeto e se o *software* resultante atende as necessidades e metas de desempenho.

Pressman (2011), descreve que um *software* não pode ser considerado de qualidade se não conseguir satisfazer a necessidade do usuário. Isto é, a qualidade de *software* ocorre quando uma gestão de qualidade efetiva é aplicada para criar um produto útil que forneça valor mensurável para aqueles que o produzem e para aqueles que o utilizam. Contudo, Pressman abre a possibilidade para que essa definição possa ser debatida, modificada e estendida.

Existem técnicas e modelos de testes que, segundo Pressman (2011), ajudam o desenvolvedor a garantir que cada produto resultante atinja suas metas de qualidade. Aplicando-se uma gama de etapas de testes, é possível, descobrir erros de lógica, manipulação de dados e comunicação entre as interfaces, e permitir que a equipe de *software* possa ajustar seu processo em caso de resultados que não atinjam as metas estabelecidas.

2.5.1 Técnicas de teste

O teste de unidade, ou unitário, de acordo com Pressman (2011), é um esforço na verificação da menor unidade do projeto (componente ou módulo). Esse método usa como guia a descrição do projeto no nível de componente, onde caminhos de controle importante são testados a fim de descobrir erros dentro dos limites do módulo. É um teste que foca na lógica interna do processamento e em estruturas de dados dentro limites do modulo testado.

O teste de caixa-preta, comportamental ou funcional é focado nos requisitos funcionais do *software*. Pressman (2011) define como uma técnica que permite derivar uma série de condições de entrada que utilizarão completamente todos os requisitos funcionais de um *software*. O teste de caixa-preta é feito na busca de encontrar: erros em funções incorretas ou faltantes, erros de interface, erros de estruturas de dados e

acesso a bases de dados externas, erros de comportamento ou desempenho, e erros na inicialização e término.

O Test Driven Development (TDD), que pode ser traduzido como Desenvolvimento Dirigidos a Testes, é baseado em pequenos ciclos de repetição, onde em cada funcionalidade do sistema é um teste criado antes de implementação. Segundo Gomes (2016), os testes são utilizados para facilitar o entendimento do projeto, clareando para o programador o que se deseja em relação ao código.

O Mock Object é utilizado para simular o comportamento de objetos em determinados cenários de uma maneira controlada. O termo Mock Object, para Medeiros (2016), é usado para descrever um caso de objetos que imitam objetos reais para teste. São muito usados com a técnica TDD e são construídos a partir de *frameworks* onde quase todo tipo de linguagem tem disponível.

2.6 Riscos

Para Macêdo (2014), os riscos são casualidades ou incidentes que, se ocorrerem, impactam diretamente um projeto. O efeito pode ser na qualidade, no escopo, no custo ou no cronograma. Essas casualidades podem ser, ou não, negativas. Os dois pontos que foram levantados como riscos críticos para o projeto são:

- Indisponibilidade do SIGA: risco de média probabilidade com alto impacto e criticidade. Nesse cenário, o projeto se tornará inacessível durante a indisponibilidade do sistema do Centro Paula Souza;
- Alteração na interface de usuário do SIGA: risco de baixa probabilidade com alto impacto e criticidade. Nessa situação, a implementação do projeto precisará ser revista para que as adequações necessárias sejam codificadas.

2.6 Cronograma

Segundo Prodanov e Freitas (2013), cronograma é a divisão em tarefas e tempo do projeto, visando o seu cumprimento e verificação do compromisso dos seus integrantes. Atividades podem ser sobrepostas sendo feitas ao mesmo tempo e existem processos que são dependentes, ficando no final de seus precedentes.

Quadro 2.1 - Cronograma

CRONOGRAMA																								
ATIVIDADES/MÊS	AGO		SET		OUT		NOV		DEZ		JAN		FEV		MAR		ABR		MAI		JUN	
Definição do tema e do orientador																						
Discussão teórica em função da determinação dos objetivos																						
Análise de viabilidade do projeto																						
Elaboração dos Pré Textuais e Introdução																						
Elaboração da Fundamentação Teórica																						
Elaboração da Metodologia																						
Revisão da redação e elementos textuais. 1ª parte																						
Entrega da versão final																						
Execução da Pré - Banca																						
Revisão Pós - Banca 1ª parte																						
Testes Durante Elaboração do Projeto																						
Elaboração do desenvolvimento (Projeto)																						
Elaboração das Considerações finais																						
Revisão da redação e elementos textuais. 2ª parte																						
Entrega da versão final do TCC																						
Execução da Banca																						
Revisão Pós - Banca 2ª parte																						

Fonte: Autoria própria (2016)

Conforme o cronograma apresentado no Quadro 2.1, foi possível facilitar a divisão de necessidades e prioridades, possibilitando a verificação dos maiores desafios e necessidades do projeto.

3. DESENVOLVIMENTO

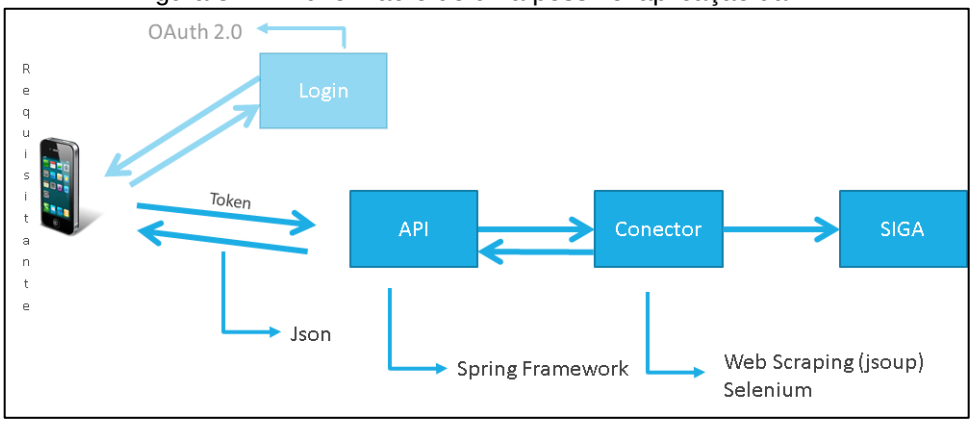
Este capítulo demonstrará as ferramentas e técnicas de programação, apresentadas no capítulo de fundamentação teórica, estruturadas para a formulação da API que receberá dados específicos do SIGA.

3.1 Estrutura e objetivo do projeto

A ideia do projeto é apresentar um arcabouço fácil e robusto que possa ser utilizada por futuros programadores. Nesse sentido, a API será estruturada para autenticar o usuário e coletar as informações solicitadas da aplicação SIGA.

Uma aplicação qualquer como, por exemplo, um aplicativo de *smartphone*, utilizando os mecanismos de autenticação demonstrados nos próximos tópicos, poderá, após obter a autorização, coletar as informações solicitadas fornecendo os parâmetros necessários para o método desejado.

Figura 3.1 – Fluxo macro de uma possível aplicação da API



Fonte: Autoria própria (2016)

Para tornar mais fácil o entendimento, a seguinte situação pode ser verificada, como exemplo, na Figura 3.1. Antes de tudo, o sistema solicitante deverá realizar a autenticação, já detalhada anteriormente. Em seguida, para obter uma nota, a aplicação deverá informar à API, utilizando a formatação JSON, o semestre e a matéria desejados. A API por sua vez, utilizará de um módulo para realizar o acesso e obtenção dos dados do SIGA. Essa coleta de dados ocorrerá através de Web Scraping, do Selenium, que simulará a navegação no SIGA (como se o próprio usuário

estivesse indo até o dado necessário) e do Jsoup que fará a extração dos dados dos HTMLs obtidos durante a navegação simulada.

A API, que será desenvolvida nas páginas seguintes, tem algumas características comuns aos *softwares* e sistemas, tais como: requisitos funcionais e não funcionais, casos de uso e sequências lógicas, porém, alguns detalhes não poderam ser demonstrados já que o mecanismo proposto agirá como uma ponte e não como um aplicativo fim. Por esse motivo, as subseções a seguir tentam se aproximar, ao máximo, os detalhes técnicos desse projeto.

3.1.1 Produto desse projeto

O produto do projeto é a criação de uma API para inspirar e facilitar a criação de aplicativos, ou sistemas, que utilizem informações do SIGA, tentando economizar tempo do desenvolvedor, de modo que não seria necessário a criação de um código para buscar e recuperar essas informações, mas, apenas, a utilização de métodos prontos. O programador, utilizando a API, pode agilizar o processo de criação e focar em outros aspectos e necessidades de seu produto.

3.1.2 Objetivo do projeto

Esse projeto tem o objetivo de disponibilizar um mecanismo para facilitar a criação de aplicativos, ou sistemas, que utilizem informações do SIGA, como, por exemplo, aplicativos para que os alunos possam acompanhar as suas faltas parciais, notas parciais e horário de suas aulas.

3.1.3 Descrição do problema - Justificativa

Atualmente, o mundo está cada vez mais conectado. O número de *smartphones* e dispositivos conectados à rede mundial está aumentando exponencialmente. Uma projeção da Fundação Getúlio Vargas de São Paulo estima que, em 2018, o número de aparelhos conectados à internet no Brasil será de dois por habitante, alcançando os 416 milhões de dispositivos (Fundação Getúlio Vargas, 2016). Desenvolver uma estrutura para facilitar a criação de sistemas de fácil acesso e ágil – onde os alunos e professores possam acompanhar as informações

acadêmicas, *websites* e formulários de inscrição dinâmicos, podem ser um grande diferencial.

O Quadro 3.1 apresenta os detalhes do problema ponderada pelos autores do projeto, os afetados, o impacto e uma possível solução, o tema dessa iniciativa: a API.

Quadro 3.1 – Quadro de problemas

O problema de:	Falta de aplicativos, *websites*, ferramentas alimentadas pelas informações acadêmicas.
Afeta:	Todo o corpo discente e docente.
Cujo impacto é:	Lentidão ou perda na transmissão de informações entre professores e alunos. Falta de automação em processos de inscrição de palestras. Falta de acompanhamento dos alunos das informações inseridas.
Uma boa solução seria:	Adaptação do sistema *on-line* para uma versão *mobile*.

Fonte: Autoria própria (2017)

O produto do projeto economizará tempo para futuros desenvolvedores, estimulando-os, assim, a desenvolver tais aplicativos.

3.1.4 Premissas iniciais

Para o desenvolvimento desse projeto, algumas das premissas abaixo são fundamentais:

- O sistema *on-line* (SIGA) se manter com o mesmo método de visualização das informações do aluno, assim como a disposição dessas informações nas páginas.
- Os sistemas on-line apresentarem alta disponibilidade para consulta das informações dos alunos.
- *Smartphones* com acesso à internet de qualquer plataforma.

3.1.5 Restrições iniciais

Por ser uma das linguagens de maior aceitação e com maior quantidade de desenvolvedores, atualmente, a aplicação será desenvolvida em Java, utilizando-se das melhores práticas.

3.1.6 Fatores de risco potenciais

Todos os itens, listados abaixo, são de alta criticidade para continuidade e entrega do projeto no prazo esperado, atrasos ou antecipações podem ocorrer, se qualquer um desses pontos apresentarem problemas:

- Recusa de participação dos discentes.
- Discentes com atitudes negativas em relação ao projeto
- Conflitos entre os membros da equipe de desenvolvimento
- Equipe de desenvolvimento não familiarizada com as ferramentas
- Membros da equipe inexperientes.
- Reestruturação do SIGA durante o projeto.
- Alto nível de complexidade técnica
- Burocracia excessiva
- Planejamento inadequado do prazo
- Baixa produtividade
- Falta de definição dos marcos do projeto
- Comunicação ineficiente

3.1.7 Prazo preliminar do projeto

A data de entrega primaria do projeto está prevista e ajustada para o dia 13 de maio de 2017. Nessa fase será necessário o envio de três (3) vias para a análise da banca e, posteriormente, avaliação.

A entrega definitiva, corrigida, e atualizada com as informações solicitadas pelos professores, componentes da banca, está prevista para 10 de junho de 2017. Nessa etapa o arquivo deverá ser entregue em formato digital.

3.1.8 Estimativa inicial de custos

Os custos estimados desse projeto foram descartados. Como o projeto não possuí um banco de dados vinculado, não foi necessário a aquisição de espaço em servidor e/ou aquisição de um host e domínio específicos para o projeto, sendo utilizadas apenas as ferramentas gratuitas existentes.

3.2 Detalhamento técnico da API

Nas subseções abaixo será possível verificar os detalhamentos técnicos da API, tais como: seus requisitos funcionais e não funcionais; casos de uso, diagrama de classes e diagrama de sequência.

3.2.1 Requisitos funcionais

Os requisitos funcionais demonstram o comportamento da API perante as entradas. No Quadro 3.2 é descrita a relação dos requisitos funcionais identificados neste projeto:

Quadro 3.2 - Quadro requisitos funcionais

Efetuar login no SIGA (RF1)	
Descrição	Efetuar login no SIGA utilizado credenciais fornecidas pelo usuário.
Exibir faltas parciais do aluno (RF2)	
Descrição	Mostrar a quantidade de faltas do aluno nas matérias.
Exibir notas parciais do aluno (RF3)	
Descrição	Recuperar e exibir as notas parciais exibidas no site.
Exibir nome (RF4)	
Descrição	Recuperar e exibir o nome do usuário que está utilizando o sistema.
Exibir Curso (RF5)	
Descrição	Recuperar e exibir o nome do curso que o usuário está cursando.
Exibir Período (RF6)	
Descrição	Recuperar e exibir o período em qual o aluno está matriculado.
Exibir Registro do aluno (RF7)	
Descrição	Recuperar e exibir o RA do aluno que está utilizando o sistema.
Exibir percentual de conclusão de curso (RF8)	
Descrição	Recuperar e mostrar qual porcentagem do curso o usuário concluiu.
Exibir percentual de rendimento (RF9)	
Descrição	Recuperar e mostrar o percentual de rendimento do aluno.
Exibir maior percentual de rendimento de curso (RF10)	
Descrição	Exibir qual o maior percentual de rendimento do curso.

Fonte: Autoria própria (2016)

Segundo Ventura (2016), quando falamos de um requisito funcional estamos nos referindo à requisição de uma função que um *software* deverá atender/realizar, ou seja, exigência, solicitação, desejo, necessidade, que um *software* materializará.

3.2.2 Requisitos não funcionais

Os requisitos não funcionais são as restrições que podem ser utilizadas para avaliar o funcionamento de um sistema. No Quadro 3.3 é descrita a relação dos requisitos não funcionais identificados neste projeto:

Quadro 3.3 – Quadro requisitos não funcionais

Documentação (RNF1)	
Descrição	A documentação *online* incluirá um manual de referência e exemplos de implementação
Adequação (RNF2)	
Descrição	O API se adequará ao processo para consultas dos dados do usaurio no SIGA.
Operacionalidade (RNF3)	
Descrição	A API atenderá as necessidades descritas pelo usuário.
Usabilidade (RNF4)	
Descrição	A API atenderá as necessidades definidas pelo desenvolvedor de forma intuitiva.
Comportamento no processo (RNF5)	
Descrição	A API atenderá a abrangência definida pelo usuário durante o mapeamento do cenário.
Performance de interface (RNF6)	
Descrição	Todas as interfaces devem atender a cada processo de maneira performática e que traga resultado.
Flexibilidade (RNF7)	
Descrição	O produto não vai se adaptar em caso de mudanças no SIGA.
Estabilidade (RNF8)	
Descrição	O produto terá que atender as necessidades de maneira estável sem perder performance.
Adaptabilidade (RNF9)	
Descrição	O produto não vai se adaptar em caso de mudanças no SIGA ou aplicação que o esteja utilizando.
Instabilidade (RNF10)	
Descrição	O produto será colocado em produção após ser publicado no servidor do cliente.
Disponibilidade (RNF11)	
Descrição	99% de disponibilidade em horário comercial
Segurança (RNF12)	
Descrição	Apenas aplicativos previamente selecionados terão acesso à API;
	Apenas usuários com o perfil aluno devem autenticar-se na API;
	O produto deve ser acessível apenas através do protocolo HTTPs;
	A autenticação será realizada através de fluxo Authorization do OAuth2, utilizando JWT assinado com mecanismo de chave pública/privada.
Utilização (RNF13)	
Descrição	A API poderá ser utilizada por até 300 usuários simultâneos.
Desempenho (RNF14)	
Descrição	O produto deve fornecer a resposta em até 30 segundos.
Portabilidade (RNF15)	
Descrição	O produto deve ser construído de forma que possa ser executado em ambiente Linux ou ambiente Windows sendo executado em Tomcat, Jetty, Jboss ou Glassfish.

Fonte: Autoria própria (2016)

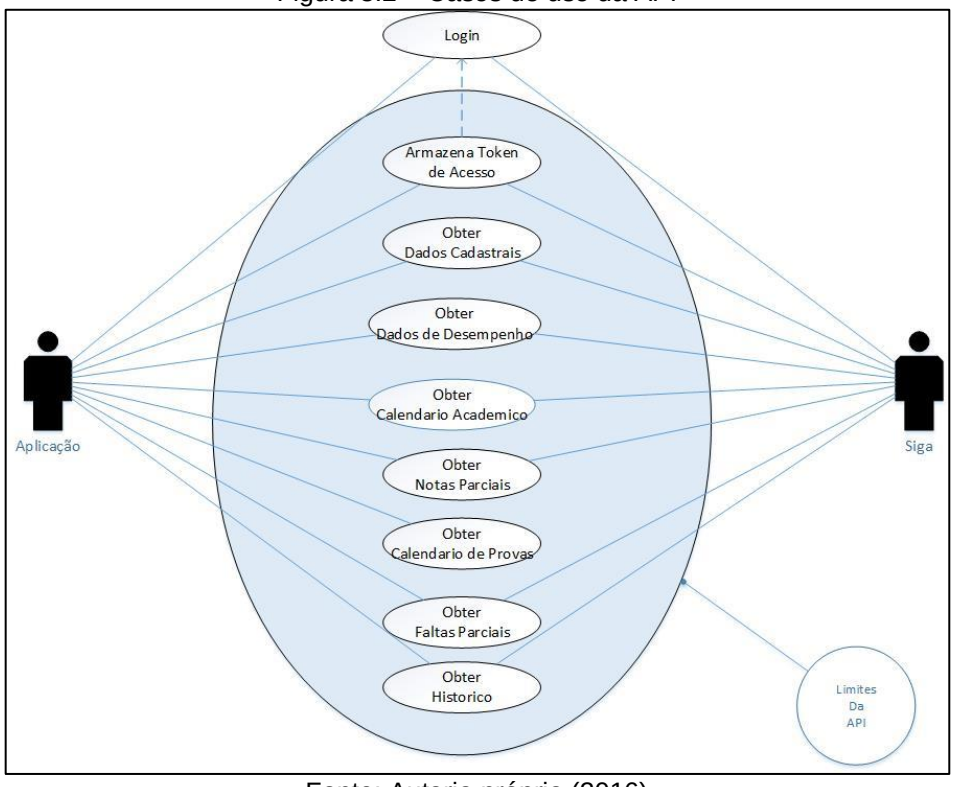
Na maior parte dos casos, esses requisitos baseiam o sistema como um todo. Isso significa que, na maioria das vezes, eles são mais importantes que os requisitos funcionais individuais. Se uma falha em cumprir um requisito funcional pode comprometer parte do sistema, uma falha em cumprir um requisito não funcional pode tornar todo o sistema inútil. (SOMMERVILLE, 2008)

3.2.3 Casos de uso

Segundo Pressman (2011), desenvolvedores e usuários podem criar cenários que identifiquem um roteiro de uso para o sistema a ser construído, os cenários normalmente são chamados de casos de uso.

Ainda segundo o autor, um caso de uso deve representar uma unidade discreta da interação entre o usuário e o sistema, por exemplo: “realizar login”, “fazer pedido”, etc. Os casos de devem ser relacionados a "atores". Um ator pode ser uma pessoa ou sistema que interage com a aplicação para executar um significante trabalho.

Figura 3.2 – Casos de uso da API



Fonte: Autoria própria (2016)

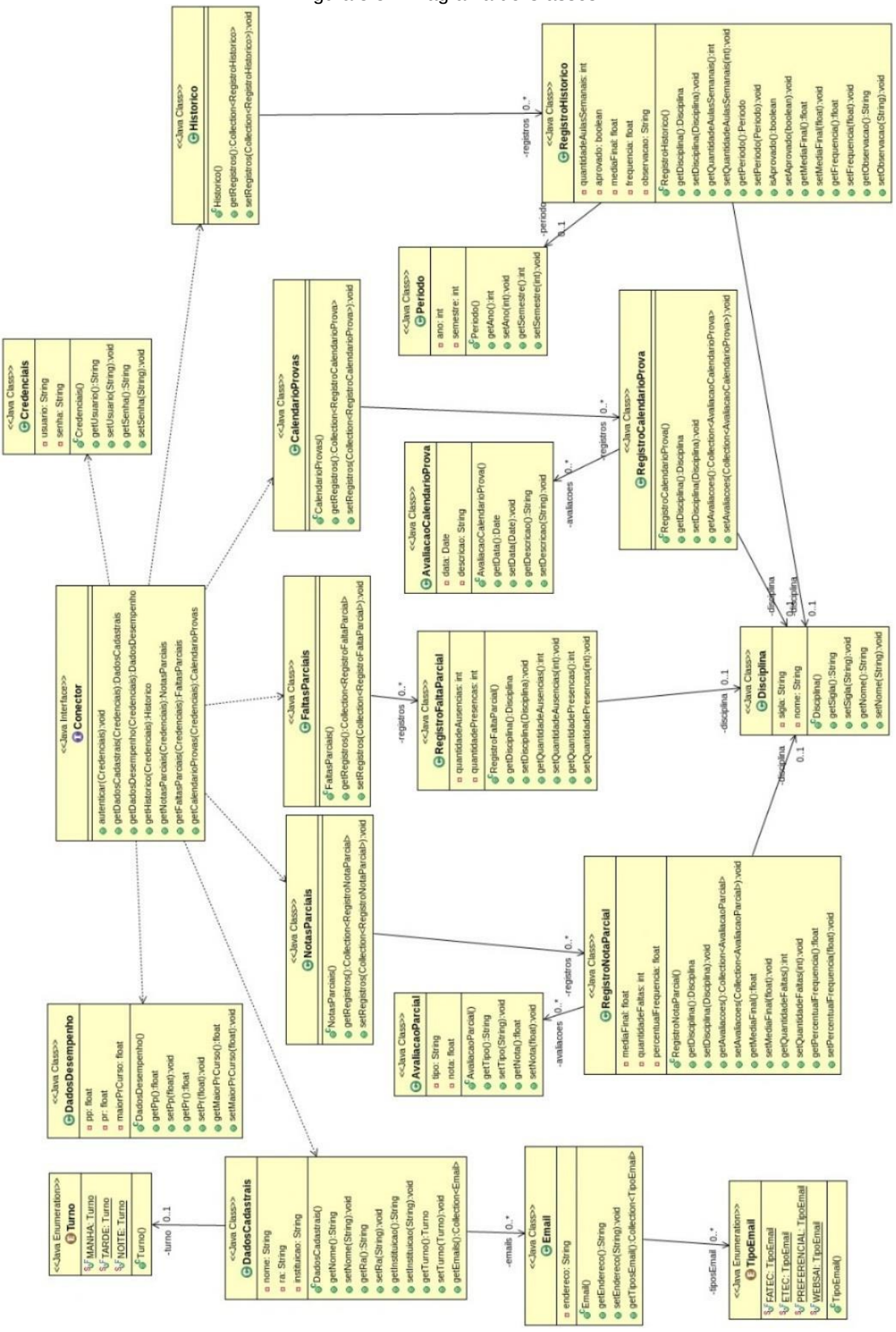
Conforme a Figura 3.2, a API possui apenas dois atores: Uma aplicação qualquer, atuando aqui como um usuário, poderá fazer apenas as seguintes consultas: dados cadastrais, desempenho, histórico, notas parciais, faltas parciais, calendário de provas e calendário acadêmico. Na outra ponta, o SIGA terá a responsabilidade de autenticar e prover à API todos os dados solicitados pela aplicação.

3.2.4 Diagrama de classes

Na programação, um diagrama de classes é uma representação do esqueleto e das relações das classes que servem de modelo para objetos. Provendo uma perspectiva estratégica, os diagramas possuem os itens de classe, atributo, operação e associação.

A Figura 3.3 representa um diagrama de classes do módulo Conector da API. Também estão representados na imagem os métodos que serão utilizados, as classes que estes métodos retornam, etc.

Figura 3.3 – Diagrama de Classes

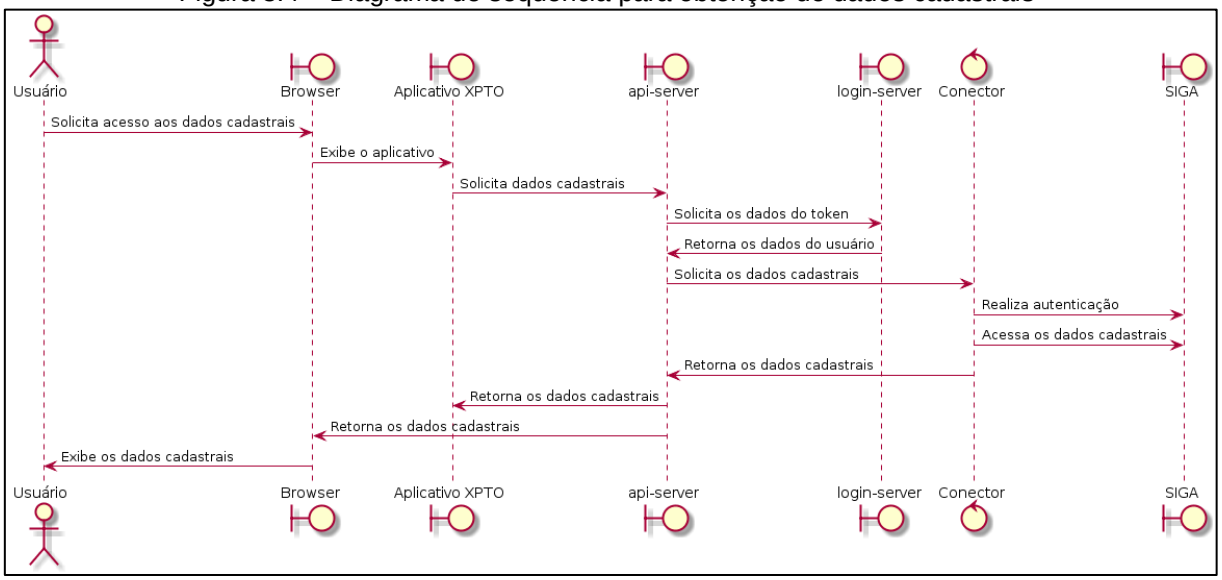


Fonte: Autoria Própria (2016)

3.2.5 Diagramas de sequência

Simboliza o processo e suas sequências, entre envios e retornos de mensagens, que são distribuídas para os objetos dentro da aplicação. É utilizado para clarificar as informações que são repassadas no diagrama de classes, pois possuí uma visualização facilitada, sem que se tenha grande conhecimento das classes e os retornos utilizados entre elas.

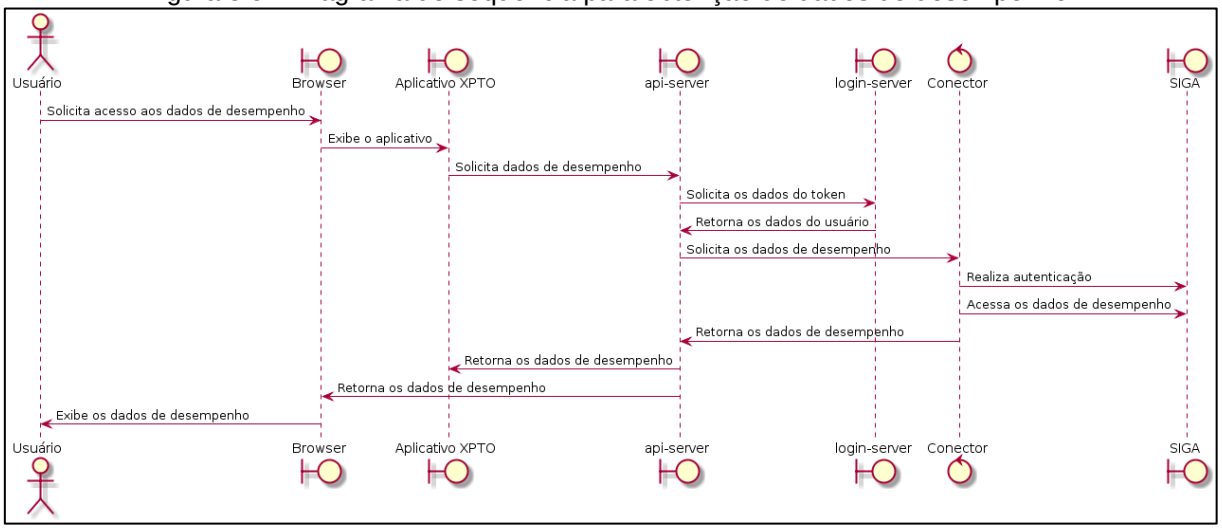
Figura 3.4 – Diagrama de sequência para obtenção de dados cadastrais



Fonte: Autoria própria (2016)

Na Figura 3.4 (obter dados cadastrais) é possível observar o diagrama de sequência para o caso de uso: obter dados cadastrais. Sendo assim, pode-se verificar, em detalhes, os papéis do usuário, dos módulos da API e do SIGA, nessa função.

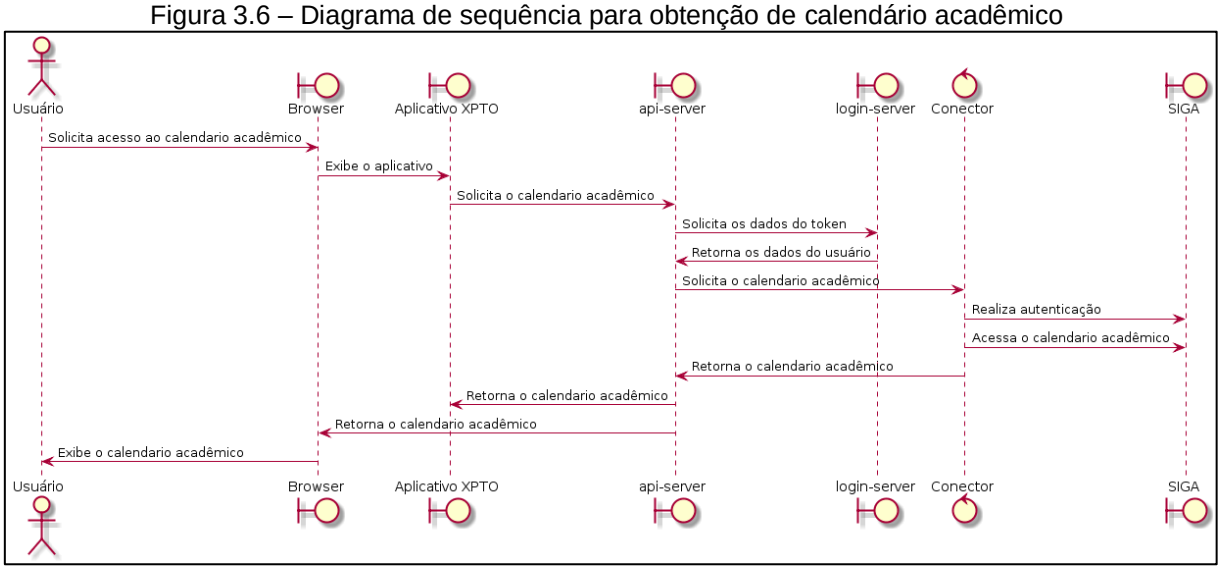
Figura 3.5 – Diagrama de sequência para obtenção de dados de desempenho



Fonte: Autoria própria (2016)

Na Figura 3.5 (obter dados de desempenho) é possível observar o diagrama de sequência para o caso de uso: obter dados de desempenho. Sendo assim, pode-se verificar, em detalhes, os papéis do usuário, dos módulos da API e do SIGA, nessa função.

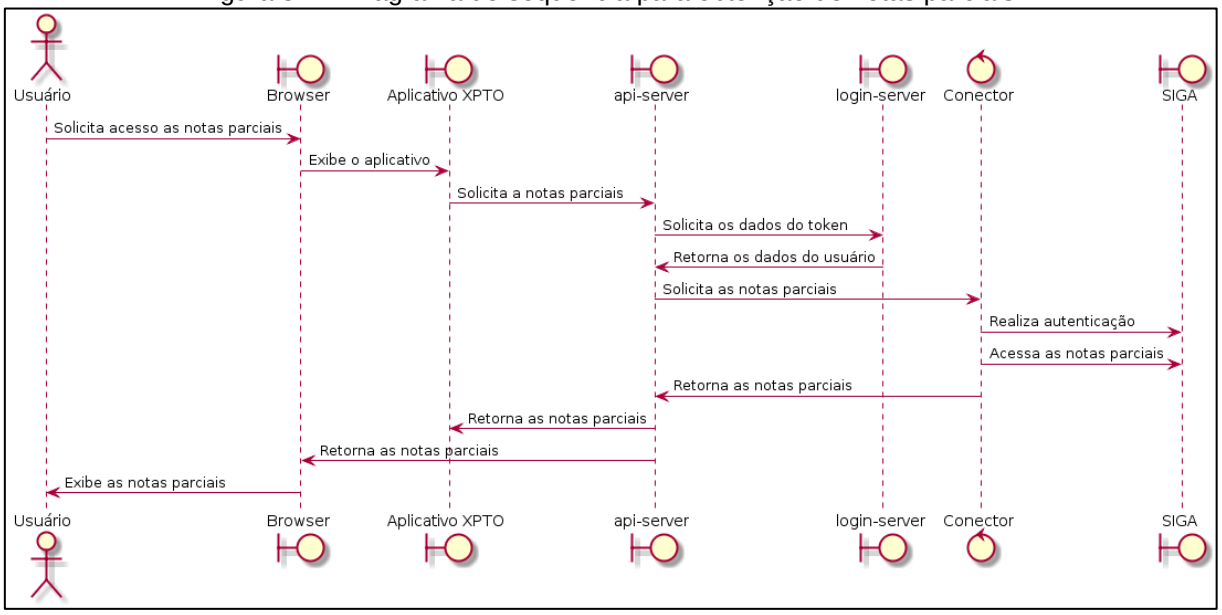
Figura 3.6 – Diagrama de sequência para obtenção de calendário acadêmico



Fonte: Autoria própria (2016)

Na Figura 3.6 (obter calendário acadêmico) é possível observar o diagrama de sequência para o caso de uso: obter calendário acadêmico. Sendo assim, pode-se verificar, em detalhes, os papéis do usuário, dos módulos da API e do SIGA, nessa função.

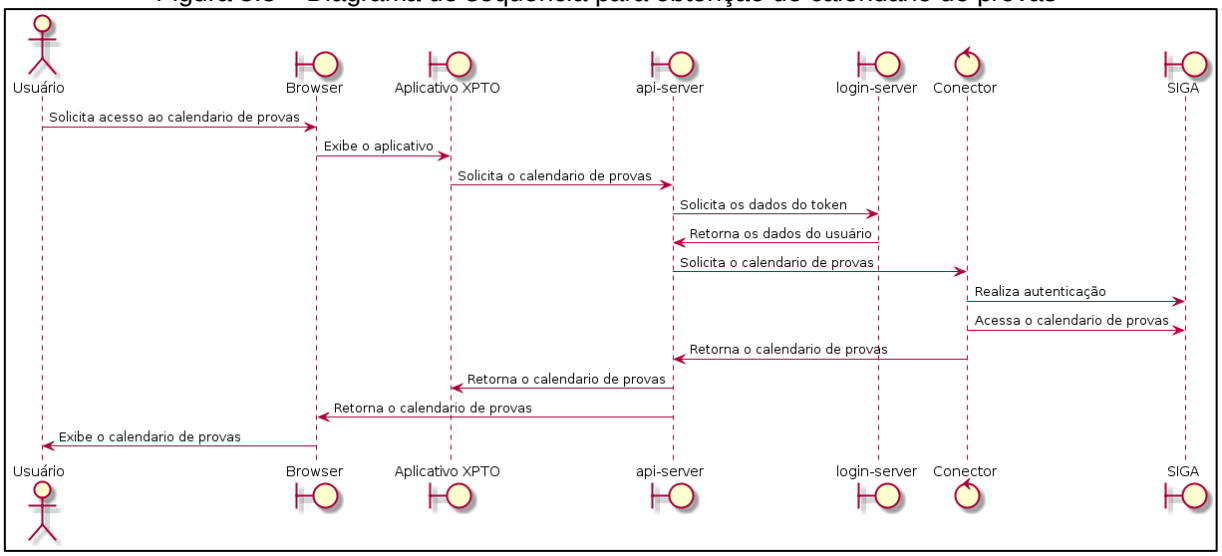
Figura 3.7 – Diagrama de sequência para obtenção de notas parciais



Fonte: Autoria própria (2016)

Na Figura 3.7 (obter notas parciais) é possível observar o diagrama de sequência para o caso de uso: obter notas parciais. Sendo assim, pode-se verificar, em detalhes, os papéis do usuário, dos módulos da API e do SIGA, nessa função.

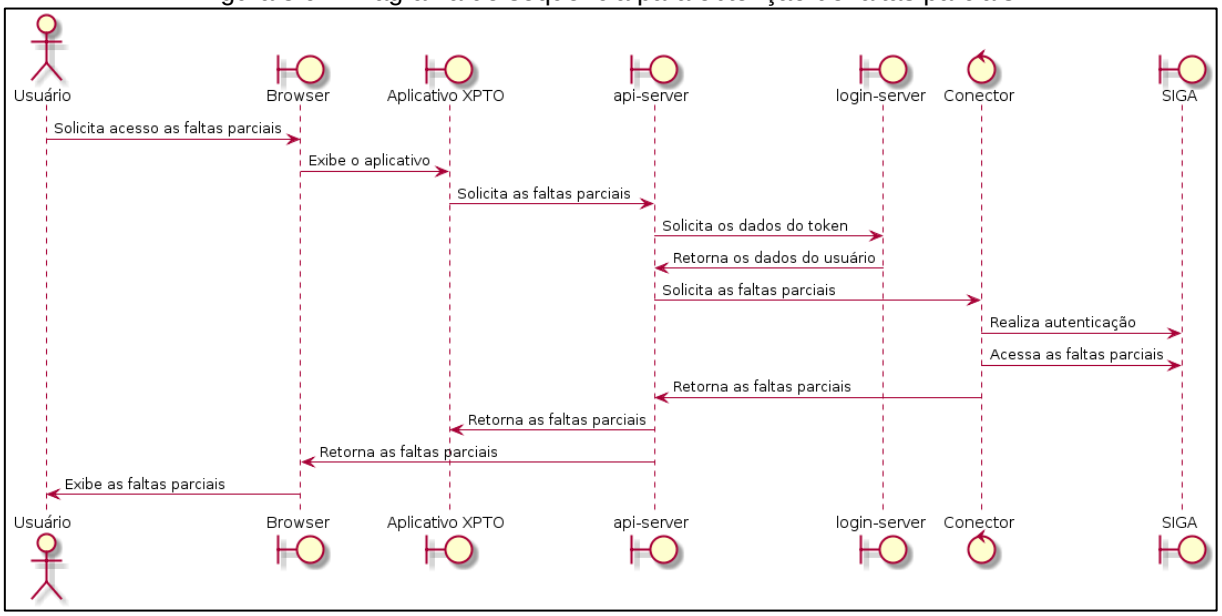
Figura 3.8 – Diagrama de sequência para obtenção de calendário de provas



Fonte: Autoria própria (2016)

Na Figura 3.8 (obter calendário de provas) é possível observar o diagrama de sequência para o caso de uso: obter calendário de provas. Sendo assim, pode-se verificar, em detalhes, os papéis do usuário, dos módulos da API e do SIGA, nessa função.

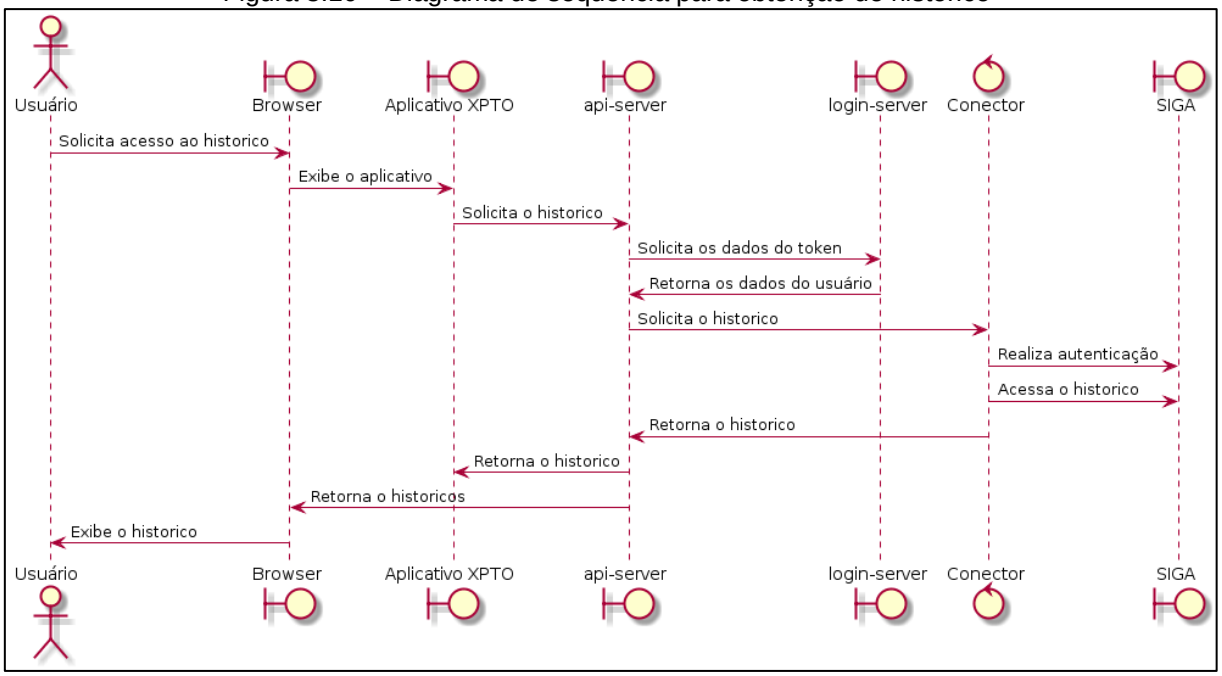
Figura 3.9 – Diagrama de sequência para obtenção de faltas parciais



Fonte: Autoria própria (2016)

Na Figura 3.9 (obter faltas parciais) é possível observar o diagrama de sequência para o caso de uso: obter faltas parciais. Sendo assim, pode-se verificar, em detalhes, os papéis do usuário, dos módulos da API e do SIGA, nessa função.

Figura 3.10 – Diagrama de sequência para obtenção de histórico



Fonte: Autoria própria (2016)

Na Figura 3.10 (obter histórico) é possível observar o diagrama de sequência para o caso de uso: obter histórico. Sendo assim, pode-se verificar, em detalhes, os papéis do usuário, dos módulos da API e do SIGA, nessa função.

3.3 Autenticação e criptografia

Uma das principais preocupações na construção desse projeto foi a questão da segurança. Durante o desenvolvimento, foi verificado que o uso do fluxo Resource Owner Password Credentials Grant apresentava uma vulnerabilidade: o fato de o cliente da API ser o responsável por obter as credenciais. Isso não garantia segurança ao usuário de que esses dados não seriam armazenados de alguma maneira.

Levando em base as considerações acima, foi prudente alterar o processo de autorização e passar a utilizar o fluxo chamado de Authorization Code Grant. Com essa técnica, a obtenção de credenciais será realizada internamente, ou seja, pelo sistema aqui apresentado, garantindo que esses dados não serão armazenados de maneira definitiva. Utilizando esse método, é possível, também, proporcionar a possibilidade de tudo ser auditado, pois o projeto tem o código aberto.

3.3.1 Classe de criptografia

Para minimizar os riscos de vulnerabilidade e as possibilidades de comprometer as chaves de criptografia, foi escolhido o método de armazenamento em memória. Um par de chaves será gerado na inicialização do módulo, com um tamanho de 2048 bits, utilizando o algoritmo RSA, de forma aleatória, gerando chaves distintas em cada execução. Na Figura 3.11 podemos ver o código responsável pela geração da chave.

Figura 3.11 – Código para geração de chave.

```
private static final String ALGORITHM = "RSA";
private static final int KEY_SIZE = 2048;

private KeyPair keyPair;

@PostConstruct
private void init() {

    KeyPairGenerator generator;
    try {
        generator = KeyPairGenerator.getInstance(ALGORITHM);
    } catch (final NoSuchAlgorithmException e) {
        log.error("Erro obtendo o KeyPairGenerator", e);
        throw new IllegalStateException(e);

    }
    generator.initialize(KEY_SIZE);

    keyPair = generator.generateKeyPair();

}
```

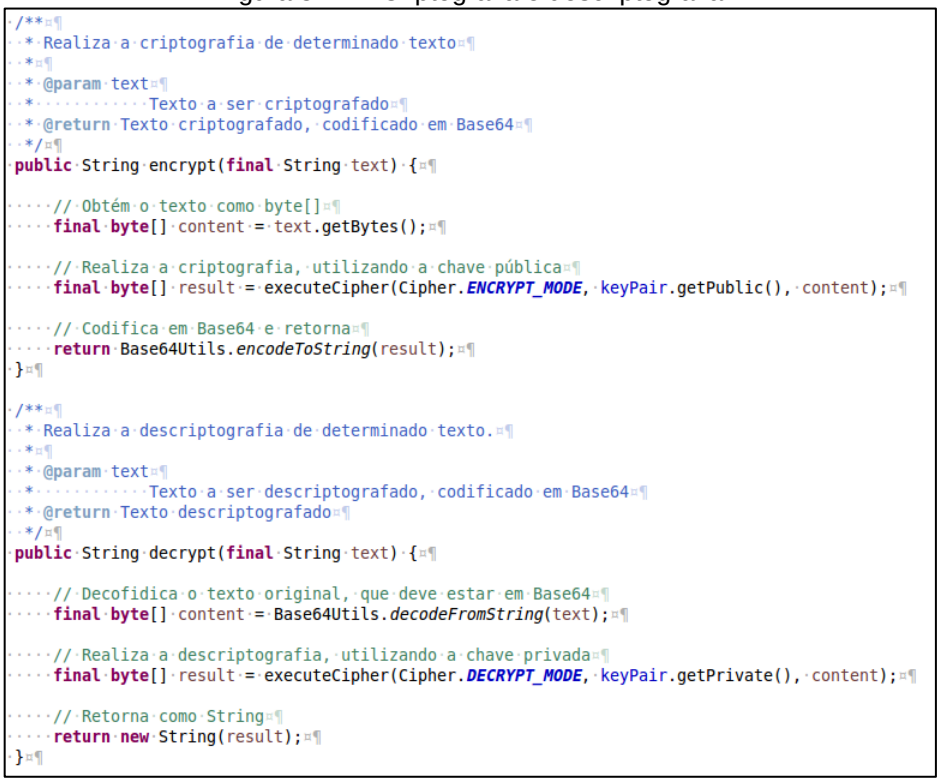
Fonte: Autoria própria (2017)

Elimina-se, através dessa abordagem, o risco associado ao fator humano. Procurou-se, também, minimizar o risco relacionado à obtenção da chave através de uma possível invasão ao sistema. O invasor precisaria ter acesso ao servidor para conseguir extrair a chave diretamente da memória. As chaves geradas sempre serão diferentes, fazendo com que os *tokens* gerados anteriormente sejam inválidos em novas utilizações.

Em caso de ataque, é possível intervir de maneira simples, apenas reiniciando o servidor. Dessa forma, será necessário que o usuário se autentique novamente, pois, como as chaves geradas sempre são diferentes, os *tokens* antigos não são aceitos.

O processo foi implementado na classe **CriptographyService**, conforme Figura 3.12, onde foram adicionados, também, os métodos responsáveis pela criptografia de textos – utilizados para armazenamento de credenciais de usuário.

Figura 3.12 – Criptografia e descriptografia

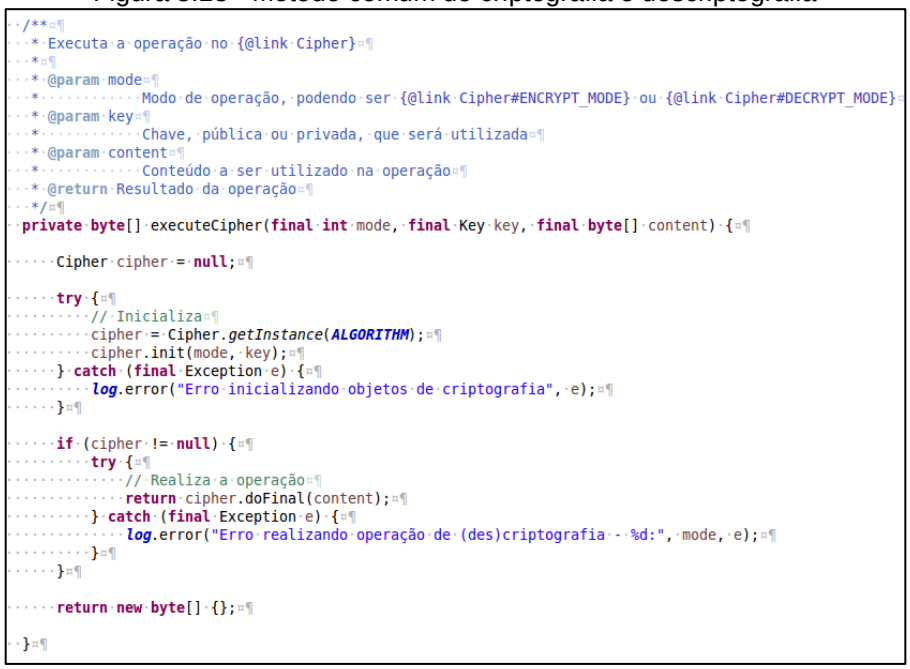


```
/**
 * Realiza a criptografia de determinado texto
 *
 * @param text
 *            Texto a ser criptografado
 * @return Texto criptografado, codificado em Base64
 */
public String encrypt(final String text) {

    // Obtém o texto como byte[]
    final byte[] content = text.getBytes();

    // Realiza a criptografia, utilizando a chave pública
    final byte[] result = executeCipher(Cipher.ENCRYPT_MODE, keyPair.getPublic(), content);

    // Codifica em Base64 e retorna
    return Base64Utils.encodeToString(result);
}

/**
 * Realiza a descriptografia de determinado texto.
 *
 * @param text
 *            Texto a ser descriptografado, codificado em Base64
 * @return Texto descriptografado
 */
public String decrypt(final String text) {

    // Decofidica o texto original, que deve estar em Base64
    final byte[] content = Base64Utils.decodeFromString(text);

    // Realiza a descriptografia, utilizando a chave privada
    final byte[] result = executeCipher(Cipher.DECRYPT_MODE, keyPair.getPrivate(), content);

    // Retorna como String
    return new String(result);
}
```

Fonte: Autoria própria (2017)

Como a realização da operação de criptografia e descriptogria, em si, possui trechos comuns, foi criado um método específico, conforme exibe a Figura 3.13, que é utilizado pelos métodos anteriores.

Figura 3.13 - Método comum de criptografia e descriptografia



```
/**
 * Executa a operação no {@link Cipher}
 *
 * @param mode
 *            Modo de operação, podendo ser {@link Cipher#ENCRYPT_MODE} ou {@link Cipher#DECRYPT_MODE}
 * @param key
 *            Chave, pública ou privada, que será utilizada
 * @param content
 *            Conteúdo a ser utilizado na operação
 * @return Resultado da operação
 */
private byte[] executeCipher(final int mode, final Key key, final byte[] content) {

    Cipher cipher = null;

    try {
        // Inicializa
        cipher = Cipher.getInstance(ALGORITHM);
        cipher.init(mode, key);
    } catch (final Exception e) {
        log.error("Erro inicializando objetos de criptografia", e);
    }

    if (cipher != null) {
        try {
            // Realiza a operação
            return cipher.doFinal(content);
        } catch (final Exception e) {
            log.error("Erro realizando operação de (des)criptografia - %d:", mode, e);
        }
    }

    return new byte[] {};

}
```

Fonte: Autoria própria (2017)

Outros métodos disponíveis nessa classe são os relacionados à conversão das chaves para *string*. Mecanismo necessário na assinatura e verificação do *token*. Nas Figuras 3.14 e 3.15, fragmentos do código dessa classe são demonstrados.

Figura 3.14 – Conversão para string

```
/**
 * Obtenção da chave privada
 *
 * @return Chave privada como {@link String}
 */
public String getPrivateKey() {

    // Obtém a chave
    final PrivateKey key = keyPair.getPrivate();
    final byte[] encoded = key.getEncoded();
    final PrivateKeyInfo pkInfo = PrivateKeyInfo.getInstance(encoded);

    ASN1Encodable encodable;
    try {
        // Obtém o conteúdo da chave
        encodable = pkInfo.parsePrivateKey();
    } catch (final IOException e) {
        log.error("Erro realizando parser da chave privada", e);
        throw new IllegalStateException(e);
    }

    // Obtém a chave como String
    return getPemKey("RSA PRIVATE KEY", encodable.toASN1Primitive());

}
```

Fonte: Autoria própria (2017)

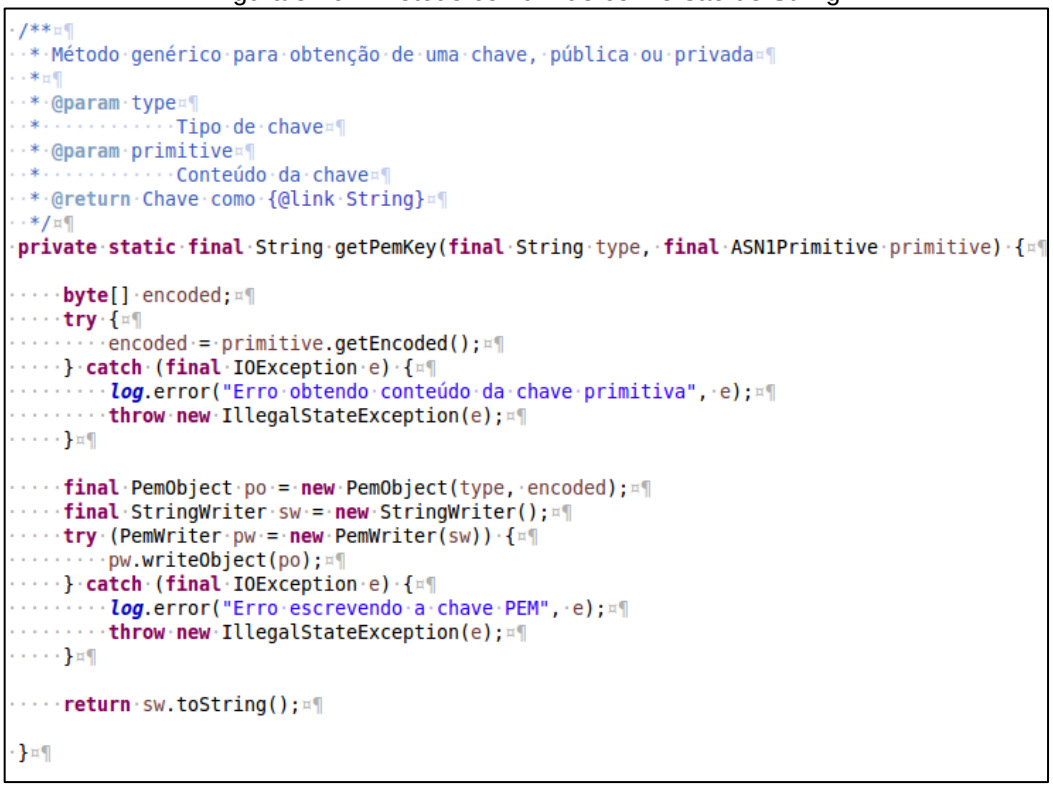
Figura 3.15 – Conversão para string (continuação)

```
/**
 * Obtenção da chave pública
 *
 * @return Chave pública, como {@link String}
 */
public String getPublicKey() {

    // Obtém a chave
    final PublicKey key = keyPair.getPublic();
    final byte[] encoded = key.getEncoded();
    final SubjectPublicKeyInfo spkInfo = SubjectPublicKeyInfo.getInstance(encoded);
    ASN1Primitive primitive;

    try {
        // Obtém o conteúdo da chave
        primitive = spkInfo.parsePublicKey();
    } catch (final IOException e) {
        log.error("Erro realizando parser da chave pública", e);
        throw new IllegalStateException(e);
    }

    // Obtém a chave como String
    return getPemKey("RSA PUBLIC KEY", primitive);

}
```

Fonte: Autoria própria (2017)

Na Figura 3.16, demonstra-se o método de conversão para *string*, unificado em apenas um método, por ser parte de uma operação comum.

Figura 3.16 – Método comum de conversão de String



```
/**
 * Método genérico para obtenção de uma chave, pública ou privada
 *
 * @param type
 *             Tipo de chave
 * @param primitive
 *             Conteúdo da chave
 * @return Chave como {@link String}
 */
private static final String getPemKey(final String type, final ASN1Primitive primitive) {

    byte[] encoded;
    try {
        encoded = primitive.getEncoded();
    } catch (final IOException e) {
        log.error("Erro obtendo conteúdo da chave primitiva", e);
        throw new IllegalStateException(e);
    }

    final PemObject po = new PemObject(type, encoded);
    final StringWriter sw = new StringWriter();
    try (PemWriter pw = new PemWriter(sw)) {
        pw.writeObject(po);
    } catch (final IOException e) {
        log.error("Erro escrevendo a chave PEM", e);
        throw new IllegalStateException(e);
    }

    return sw.toString();

}
```

Fonte: Autoria própria (2017)

3.3.2 Authorization Code Grant

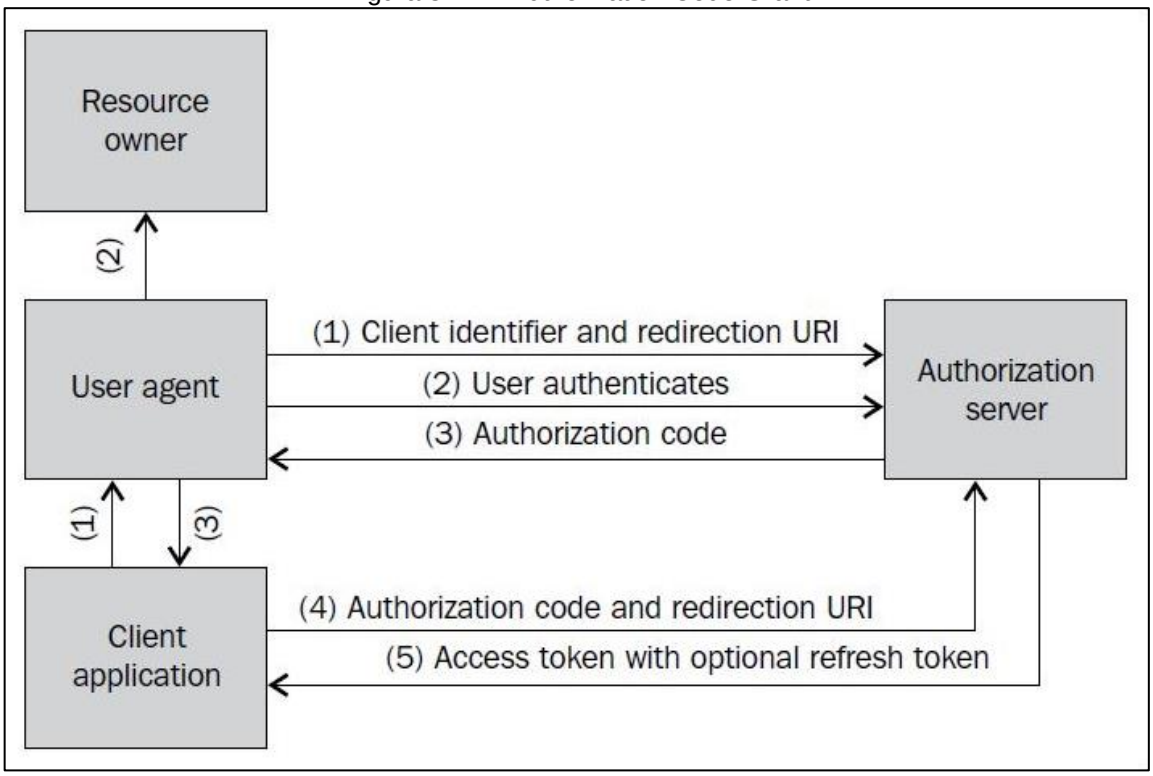
O fluxo de Authorization Code Grant, segundo Spasovski (2013, tradução própria), é utilizado para aplicações *web* em geral e é o mais escolhido, na maioria das vezes, pelos desenvolvedores. Esse mecanismo, assim como o Resource Owner, também é utilizado para realizar autenticações sem a necessidade de ter uma conta criada no ambiente em que se deseja acessar. Esse fluxo permite que o usuário acesse utilizando contas de aplicações como: Facebook, LinkedIn ou Google.

Em seu texto Spasovski (2013, tradução própria) destaca três características principais, sendo elas:

- O código de autorização é usado por clientes confidencias;
- Usa redirecionamento, portanto, é um fluxo baseado em redirecionamento, entre o cliente e o serviço de autorização;
- Requer aprovação do usuário final para autorização, exibindo, por exemplo, uma mensagem em um navegador da *web*.

O diagrama da Figura 3.17 demonstra, graficamente, uma representação do fluxo, através das especificações da OAuth 2.0.

Figura 3.17 – Authorization Code Grant



Fonte: Spasovski, 2013, p30.

Spasovski (2013, tradução própria), divide o fluxo em duas partes, a primeira é solicitação de autorização do usuário para obter o código de autorização e, depois de autorizado, a aplicação solicita um *token*, utilizando o código de autorização obtido pelo usuário.

3.3.3 Configuração dos arquivos de autenticação

A autenticação é realizada utilizando-se de dois conjuntos de credenciais, as credenciais de aplicativos clientes – aqueles autorizados a utilizar a API – e as credenciais de usuário – os alunos, que são os utilizadores dos aplicativos clientes.

Figura 3.18 – Exemplo de trecho de arquivo de configuração

```
sigapi:
  clients:
    - id: api
      secret: secret
      authority: server
      grant-types:
        - client_credentials
      token-validity: 18000
    - id: exemplo
      secret: exemplo
      authority: client
      grant-types:
        - authorization_code
      token-validity: 1800
```

Fonte: Autoria própria (2017)

A configuração dos aplicativos credenciais é realizada através da carga de um arquivo de configuração, exemplificado, em fração, na Figura 3.18. Esse arquivo, por sua vez, deve ser utilizado conforme demonstra a Figura 3.19.

Figura 3.19 – Configuração dos aplicativos

```
@Override
public void configure(final ClientDetailsServiceConfigurer configurer) throws Exception {

    final InMemoryClientDetailsServiceBuilder builder = configurer.inMemory();

    final List<Client> clients = config.getClients();
    for (final Client client : clients) {

        final String secret = config.getSecret(client);

        // @formatter:off
        builder
            .withClient(client.getId())
            .secret(secret)
            .authorities(client.getAuthority())
            .scopes("read")
            .authorizedGrantTypes(client.getGrantTypes().stream().toArray(String[]::new));
        // @formatter:on

    }

}
```

Fonte: Autoria própria (2017)

Nas Figuras 3.18 e 3.19, é possível observar dois aplicativos clientes cadastrados, cada um com um identificador (ID), senha (*secret*), permissão (*authority*), fluxos Oauth2 permitidos (*grant-types*) e tempo de validade do *token* em segundos (*token-validity*). Existem apenas dois tipos de permissão: *server* e *client*. A primeira é de uso restrito ao módulo da API. O escopo (*scope*) é definido como padrão para todos os clientes por não ser relevante na execução atual.

As credenciais do usuário, armazenadas no *token* JWT, são obtidas no momento em que é necessário realizar a autenticação. Esse processo é feito apenas no módulo de login e, utilizando o par de chaves público/privado criados, todos os dados são criptografados.

A manipulação do *token* JWT é realizada através da extensão da classe **JwtAccessTokenConverter** e possui três pontos princpais:

- A inicialização;
- O processo de adição de informações extras;
- O processo de obtenção das informações.

O processo de inicialização consiste em configurar as chaves públicas e privadas para a geração do *token*, conforme demonstrando na Figura 3.20.

Figura 3.20 – Processo de inicialização do token

```
@PostConstruct
private void init() {

    final String publicKey = criptographyService.getPublicKey();
    final String privateKey = criptographyService.getPrivateKey();

    setSigningKey(privateKey);
    setVerifierKey(publicKey);

}
```

Fonte: Autoria própria (2017)

A Figura 3.21 mostra o processo de edição de informações, que salva as credenciais do usuário no *token*, após sua criptografia.

Figura 3.21 – Adição de informações

```
@Override
public OAuth2AccessToken enhance(final OAuth2AccessToken accessToken, final OAuth2Authentication authentication) {

    final CustomUser sigaUser = (CustomUser) authentication.getPrincipal();

    // Obtém as credenciais
    final String usuario = sigaUser.getUsuario();
    final String senha = sigaUser.getPassword();

    // Realiza a criptografia
    final String usuarioEncrypted = criptographyService.encrypt(usuario);
    final String senhaEncrypted = criptographyService.encrypt(senha);

    // Cria o mapa com as informações para o token
    final Map<String, Object> information = new HashMap<>();
    information.put(ATRIBUTO_USUARIO, usuarioEncrypted);
    information.put(ATRIBUTO_SENHA, senhaEncrypted);

    // Adiciona as informações ao token
    ((DefaultOAuth2AccessToken) accessToken).setAdditionalInformation(information);

    return super.enhance(accessToken, authentication);
}
```

Fonte: Autoria própria (2017)

O processo de obtenção das informações, inversamente do que é feito no processo anterior, consiste em trabalhar os dados criptografados no texto e os retornar sem criptografia, conforme demonstrado na Figura 3.22.

Figura 3.22 – Obtenção de Informação

```
public Map<String, ?> convertAccessToken(final OAuth2AccessToken token, final OAuth2Authentication authentication) {

    // Obtém as informações do token
    final Map<String, Object> information = ((DefaultOAuth2AccessToken) token).getAdditionalInformation();

    final String usuarioEncrypted = (String) information.get(ATRIBUTO_USUARIO);
    final String senhaEncrypted = (String) information.get(ATRIBUTO_SENHA);

    // Realiza a descriptografia
    final String usuario = criptographyService.decrypt(usuarioEncrypted);
    final String senha = criptographyService.decrypt(senhaEncrypted);

    // Adiciona as informações ao mapa de retorno
    final Map<String, Object> result = (Map<String, Object>) super.convertAccessToken(token, authentication);
    result.put(ATRIBUTO_USUARIO, usuario);
    result.put(ATRIBUTO_SENHA, senha);

    return result;

}
```

Fonte: Autoria própria (2017)

A obtenção dos dados é restrita, apenas, ao módulo da API.

Figura 3.23 – Restrição ao módulo de login

```
@Override
public void configure(final AuthorizationServerSecurityConfigurer oauthServer) throws Exception {
    oauthServer.checkTokenAccess("hasAuthority('server')");
}
```

Fonte: Autoria própria (2017)

Dessa forma, somente aplicativos cadastrados com a permissão server tem acesso a esse método, conforme exibido no trecho de código da Figura 3.23

3.4 Implementação do Conector

O Conector é o elemento chave para o funcionamento da API, ele é definido através de uma interface, tornando-se, assim, parte importante no uso dos conceitos de injeção de dependência e inversão de controle.

Figura 3.24 – Interface do conector

```
package br.edu.fatecsbc.sigapi.conector;

import br.edu.fatecsbc.sigapi.conector.dto.CalendarioProvas;

public interface Conector<T extends Credenciais> {

    CredenciaisBuilder<T> getCredenciaisBuilder();

    boolean autenticar(T credenciais);

    DadosCadastrais getDadosCadastrais(T credenciais);

    DadosDesempenho getDadosDesempenho(T credenciais);

    Historico getHistorico(T credenciais);

    NotasParciais getNotasParciais(T credenciais);

    FaltasParciais getFaltasParciais(T credenciais);

    CalendarioProvas getCalendarioProvas(T credenciais);

}
```

Fonte: Autoria própria (2017)

A partir de sua definição representada pela Figura 3.24, é possível ter inúmeras implementações distintas, dando a possibilidade de que outros desenvolvedores elaborem suas próprias programações e substituam o conector existente.

3.4.1 Conector Selenium

Para a implementação desse projeto, para realizar a navegação através do SIGA, utilizou-se a ferramenta Selenium. Que tem a capacidade de simular o funcionamento de um *browser*. Após a navegação e interação com os dados de cada página, utilizando-se a biblioteca Jsoup, o código fonte é obtido e o *parser* é realizado.

A classe principal é a chamada **ConectorSelenium** e, conforme demonstrado na Figura 3.25, ela implementa a interface Conector e utiliza as classes de serviços, delegando a obtenção de cada dado solicitado.

Figura 3.25 – Trecho de código da classe ConectorSelenium

```
@Component
public class ConectorSelenium
    implements Conector<CredenciaisSelenium> {

    @Autowired
    private AutenticacaoService autenticacaoService;

    @Autowired
    private DadosCadastraisService dadosCadastraisService;

    @Autowired
    private DadosDesempenhoService dadosDesempenhoService;

    @Autowired
    private HistoricoService historicoService;

    @Autowired
    private NotasParciaisService notasParciaisService;

    @Autowired
    private FaltasParciaisService faltasParciaisService;

    @Autowired
    private CalendarioProvasService calendarioProvasService;

    @Override
    public CredenciaisBuilder<CredenciaisSelenium> getCredenciaisBuilder() {
        return new CredenciaisSeleniumBuilder();
    }

    @Override
    public boolean autenticar(final CredenciaisSelenium credenciais) {
        return autenticacaoService.autenticar(credenciais);
    }

    @Override
    public DadosCadastrais getDadosCadastrais(final CredenciaisSelenium credenciais) {
        return dadosCadastraisService.getDados(credenciais);
    }

    @Override
    public DadosDesempenho getDadosDesempenho(final CredenciaisSelenium credenciais) {
        return dadosDesempenhoService.getDados(credenciais);
    }

    @Override
    public Historico getHistorico(final CredenciaisSelenium credenciais) {
        return historicoService.getDados(credenciais);
    }
```

Fonte: Autoria própria (2017)

Essa separação foi feita para que fosse possível a alta coesão, ou seja, fazer com que cada classe cumprisse apenas um objetivo. As classes abaixo foram chamadas de classes de serviços:

- AutenticacaoService: Responsável pelo Login;
- CalendarioProvasService: Responsável pela obtenção do calendário de provas (não implementado);
- DadosCadastraisService: Responsável pela obtenção dos dados cadastrais;
- DadosDesempenhoService: Responsável pela obtenção dos dados de desempenho;
- FaltasParciaisService: Responsável pela obtenção dos dados de faltas parciais;

- HistoricoService: Responsável pela obtenção do histórico (não implementado);
- NotasParciaisService: Responsável pela obtenção das notas parciais.

Na Figura 3.26 é possível verificar um trecho da classe DadosDesempenhoService, como exemplo de uma classe de serviço.

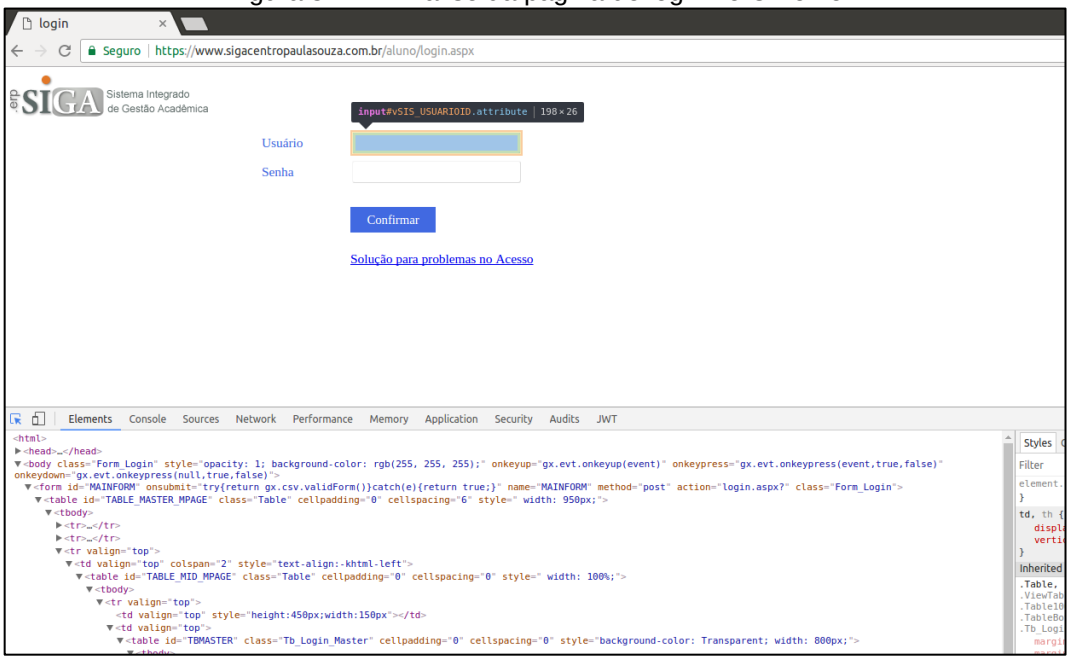
Figura 3.26 – Trecho do DadosDesempenhoService, exemplificando uma classe de serviço

```
@Service
public class DadosDesempenhoService
    extends AbstractPaginaSimplesService<DadosDesempenho> {

    @Override
    protected PaginaEnum getPagina() {
        return PaginaEnum.AVISOS;
    }

    @Override
    protected DadosDesempenho extrair(final JsoupHelper helper) {

        // PP
        final float pp = helper.getTextAsFloatFromElementById("span_MPW0039vACD_ALUNOCURSOINDICEPP");

        // PR
        final float pr = helper.getTextAsFloatFromElementById("span_MPW0039vACD_ALUNOCURSOINDICEPR");

        // Maior PR do Curso
        final float maiorPrCurso = helper.getTextAsFloatFromElementById("span_MPW0039vMAX_ACD_ALUNOCURSOINDICEPR");

        final DadosDesempenho dados = new DadosDesempenho();

        dados.setPp(pp);
        dados.setPr(pr);
        dados.setMaiorPrCurso(maiorPrCurso);

        return dados;

    }

}
```

Fonte: Autoria própria (2017)

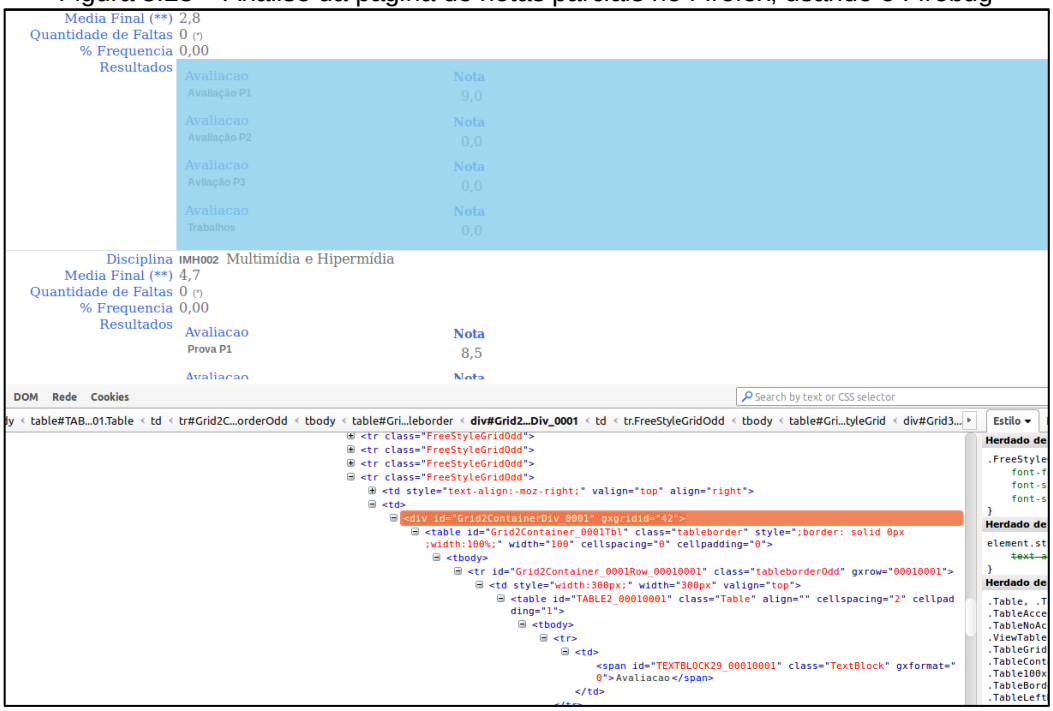
Cada serviço foi criado após análise do código HTML. Acessando o SIGA através de um *browser* convencional, como Firefox ou Chrome, foi possível extrair os elementos através dos seus IDs, nomes ou classes CSS. Nas Figuras 3.27 e 3.28, será possível encontrar a demonstração dessas análises realizadas nas páginas dos navegadores.

Figura 3.27 – Análise da página de login no Chrome



Fonte: Autoria própria (2017)

Figura 3.28 – Análise da página de notas parciais no Firefox, usando o Firebug



Fonte: Autoria própria (2017)

A classe chamada **SigaClient**, é responsável pela conexão ao SIGA. Ela é uma extensão do JBrowserDriver, componente que identifica o *browser* utilizado pelo Selenium, e é utilizada diretamente pelos serviços. Dessa forma, o controle de navegação, realização de cliques e a verificação de carregamento do conteúdo, ficam

sob responsabilidade dessa classe. Um trecho do código dessa classe pode ser visto na Figura 3.29.

Figura 3.29 – Trecho do código da classe SigaClient

```
      }¤¶

      if (newPage.isFrame()) {¤¶

          try {¤¶
              wait(ExpectedConditions.frameToBeAvailableAndSwitchToIt(newPage.getFrame()));¤¶
          } catch (final TempoEsgotadoException e) {¤¶
              throw new SigaInacessivelException(¤¶
                      "Erro inesperado tentando obter o conteúdo de um frame da página " + newPage.name(), e);¤¶
          }¤¶

      }¤¶

      currentPage = newPage;¤¶

  }¤¶

  public String goAndGetSourcePage(final PaginaEnum pagina) {¤¶

      try {¤¶
          go(pagina);¤¶
          return getPageSource();¤¶
      } catch (final SigaInacessivelException e) {¤¶
          logger.warn("Não foi possível obter o código fonte da página {}", pagina.name(), e);¤¶
      }¤¶

      return null;¤¶

  }¤¶

  public void wait(final ExpectedCondition<?> condition) throws TempoEsgotadoException {¤¶
      wait(condition, DEFAULT_TIMEOUT);¤¶
  }¤¶

  public void wait(final ExpectedCondition<?> condition, final int timeout) throws TempoEsgotadoException {¤¶

      final WebDriverWait wait = new WebDriverWait(this, timeout);¤¶
      turnOffImplicitWaits();¤¶
      try {¤¶
          wait.until(condition);¤¶
      } catch (final Exception e) {¤¶
          throw new TempoEsgotadoException(e);¤¶
      } finally {¤¶
          turnOnImplicitWaits();¤¶
      }¤¶

  }¤¶

  public void sendKeysElementById(final String id, final String keys) {¤¶
      findElement(By.id(id)).sendKeys(keys);¤¶
  }¤¶
```

Fonte: Autoria própria (2017)

Para organizar e reutilizar o código comum, foi desenvolvida a classe abstrata **AbstractService**. Conforme pode ser visto na Figura 3.30, essa classe serve como base para executar as operações comuns a todos os serviços, como a realização da autenticação, e define um método que todas as classes devem implementar: o **getDados**.

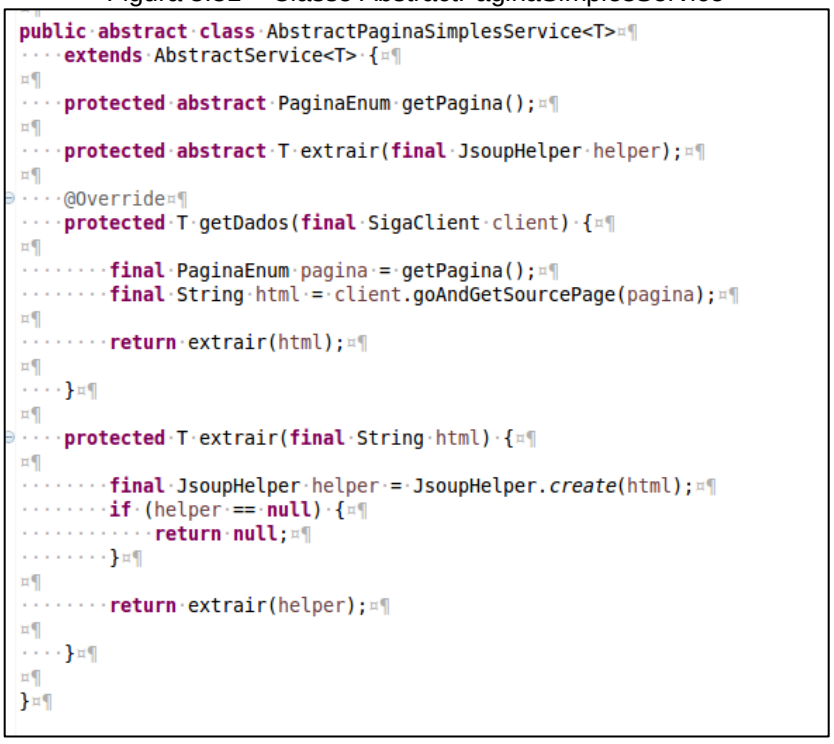
Figura 3.30 – Classe AbstractService

```
public abstract class AbstractService<T> {

    protected Logger logger = LoggerFactory.getLogger(this.getClass());

    @Autowired
    private AutenticacaoService autenticacaoService;

    protected abstract T getDados(SigaClient client);

    public T getDados(final CredenciaisSelenium credenciais) {

        SigaClient client = null;
        try {
            client = autenticacaoService.conectar(credenciais);
            if (client != null) {
                return getDados(client);
            }
        } finally {
            if (client != null) {
                client.quit();
            }
        }

        return null;

    }

}
```

Fonte: Autoria própria (2017)

Outra classe importante, destacada na Figura 3.31, é a **AbstractPaginaSimplesService**, que também é utilizada como base, mas apenas para dados que obtenham conteúdo de apenas uma página. Esse mecanismo foi implementado em todas as páginas, exceto pela obtenção dos dados cadastrais.

Figura 3.31 – Classe AbstractPaginaSimplesService



```
public abstract class AbstractPaginaSimplesService<T>
    extends AbstractService<T> {

    protected abstract PaginaEnum getPagina();

    protected abstract T extrair(final JsoupHelper helper);

    @Override
    protected T getDados(final SigaClient client) {

        final PaginaEnum pagina = getPagina();
        final String html = client.goAndGetSourcePage(pagina);

        return extrair(html);

    }

    protected T extrair(final String html) {

        final JsoupHelper helper = JsoupHelper.create(html);
        if (helper == null) {
            return null;
        }

        return extrair(helper);

    }

}
```

Fonte: Autoria própria (2017)

Foi necessária a criação de um mecanismo de exceção para obter os e-mails, que fazem parte dos dados cadastrais, pois é uma informação contida em um iframe, ao contrário de todos os outros dados que estão diretamente na página. Na Figura 3.32 é o trecho da classe **DadosCadastraisService**, utilizada devido a necessidade de navegar por duas páginas para obter os dados cadastrais e e-mails

Figura 3.32 – Trecho da classe DadosCadastraisService

```
@Component
public class DadosCadastraisService
    extends AbstractService<DadosCadastrais> {

    @Override
    protected DadosCadastrais getDados(final SigaClient client) {

        // Obtém o HTML da página de avisos (que contém os dados cadastrais)
        final String htmlAvisos = client.goAndGetSourcePage(PaginaEnum.AVISOS);

        // Obtém o HTML da página de emails
        final String htmlEmails = client.goAndGetSourcePage(PaginaEnum.EMAILS);

        // Caso tenha falhado em pegar as duas, houve algum problema, então o retorno precisa ser nulo
        if (StringUtils.isBlank(htmlAvisos) && StringUtils.isBlank(htmlEmails)) {
            return null;
        }

        return extrair(htmlAvisos, htmlEmails);

    }

    protected DadosCadastrais extrair(final String htmlAvisos, final String htmlEmails) {

        // Extrai e retorna os dados
        final DadosCadastrais dadosCadastrais = new DadosCadastrais();
        extrairDadosCadastrais(dadosCadastrais, htmlAvisos);
        extrairEmails(dadosCadastrais, htmlEmails);
        return dadosCadastrais;

    }

    private void extrairDadosCadastrais(final DadosCadastrais dadosCadastrais, final String html) {

        final JsoupHelper helper = JsoupHelper.create(html);

        if (helper == null) {
            return;
        }

        final String nome = helper.getTextFromElementById("span_MPW0039vPRO_PESSOALNOME", true);
        final String foto = helper.getImageSrcById("MPW0039FOTO");
        final String ra = helper.getTextFromElementById("span_MPW0039vACD_ALUNOCURSOREGISTROACADEMICOCURSO");
        final String instituicao = helper.getTextFromElementById("span_vUNI_UNIDADENOME_MPAGE", true);
        final String curso = helper.getTextFromElementById("span_vACD_CURSONOME_MPAGE", true);
        final String periodo = helper.getTextFromElementById("span_vACD_PERIODODESCRICAO_MPAGE");
```

Fonte: Autoria própria (2017)

Outra ação necessária foi a padronização de alguns campos a fim de criar um comportamento previsível. Por exemplo: o campo de nomes de avaliações é descrito utilizando diferentes nomenclaturas, como: “P1”, “Avaliação P1”, “Avaliação Oficial P1” e assim por diante. Nesses casos foi implementado uma rotina para sempre retornar o dado utilizando uma nomenclatura única.

Figura 3.33 – Método utilizado para padronizar o retorno de provas e trabalhos

```
private static final String ajustaTipo(final String s) {

    final String tmp = StringUtils.trimToNull(s);
    if (tmp != null) {

        final char last = tmp.charAt(tmp.length() - 1);

        // Provas
        if (Arrays.asList('1', '2', '3').contains(last) && StringUtils.containsIgnoreCase(tmp, "P" + last)) {
            return "Avaliação Oficial - P" + last;
        }

        // Trabalhos
        if (Arrays.asList("trabalho", "trabalhos", "avaliação t").contains(tmp.toLowerCase())) {
            return "Trabalho";
        }

        return StringHelper.capitalize(tmp);

    }

    return null;

}
```

Fonte: Autoria própria (2017)

Na Figura 3.33 é possível verificar o método criado para padronizar o nome utilizado pelas avaliações no SIGA.

3.5 Elaboração dos módulos

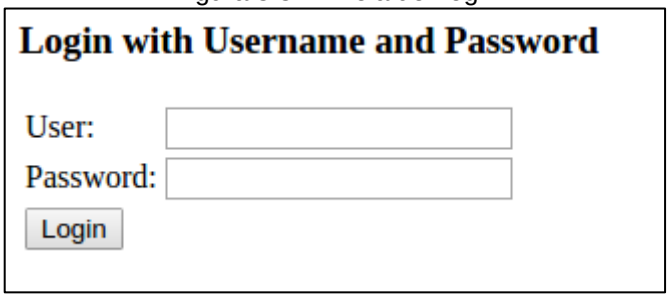
Os módulos são os conjuntos de classes e configurações do *Spring Framework*. Esses são os principais artefatos do projeto e é constituído por dois módulos: o módulo de Login e o módulo da API.

3.5.1 Módulo de login

A criação do módulo de login consistiu-se na configuração de classes, utilizando o *Spring Security*, para realizar todo o controle de acesso e geração do *token* JWT, que utilizará um par de chaves pública/privada, esse módulo é responsável por todo o processo de autorização e autenticação do usuário.

O uso desse módulo é visível ao usuário final, pois ele é redirecionado para tela de login, onde suas credenciais são solicitadas e, caso haja êxito na operação, é redirecionado para a tela de autorização, as telas não foram customizadas porque não era o objetivo principal desse projeto, conforme pode ser visto na Figura 3.34 onde é apresentada a tela de login.

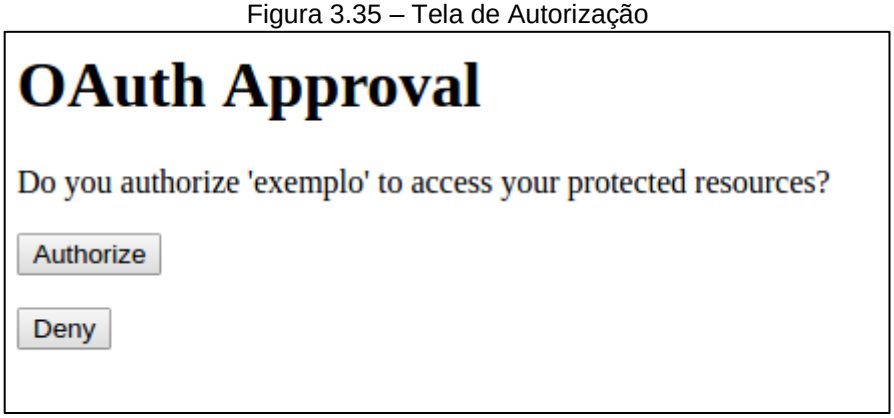
Figura 3.34 – Tela de Login



Fonte: Autoria própria (2017)

A Figura 3.35 mostra a tela que apresenta a requisição de autorização por parte do usuário para o acesso do OAuth aos dados desse.

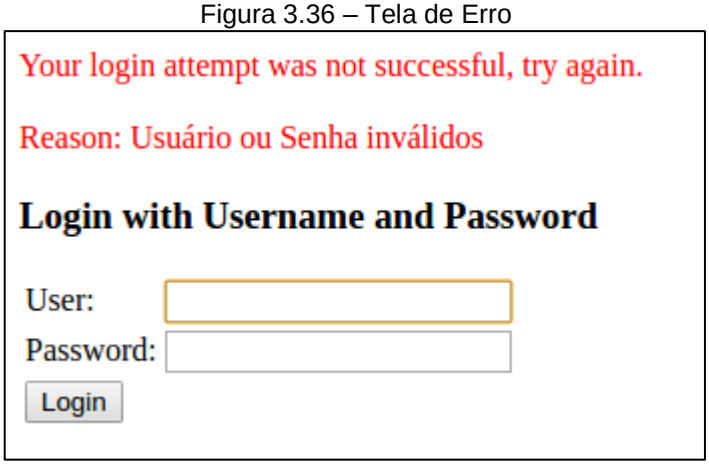
Figura 3.35 – Tela de Autorização



Fonte: Autoria própria (2017)

É possível ver, na Figura 3.36, o erro exibido caso o usuário cometa algum engano ao digitar o login e senha. O sistema retornará o erro de autorização.

Figura 3.36 – Tela de Erro



Fonte: Autoria própria (2017)

O fluxo de login mantém um cache de 30 minutos, permitindo que posteriores acessos sejam feitos por parte do mesmo usuário. Isso foi feito para que os demais acessos no mesmo período ocorram de maneira mais rápida.

Após a autorização, o fluxo de acesso é redirecionado novamente para a aplicação cliente, informando um código de autorização. De posse desse código temporário de autorização (temporário e possuí uma curta duração), a própria aplicação faz a solicitação e troca dessa combinação por um *token* de acesso, que possuí um tempo de validade maior, porém, variando da aplicação que a utiliza. Após a obtenção do *token* de acesso, o código de autorização se torna inválido.

Na Figura 3.37 é possível ver o retorno da chamada ao serviço para obtenção do *token*.

Figura 3.37 – Retorno do serviço de obtenção de token


```
{
  "jti": "e8339c42-0328-48ea-8f86-cf7315c8755c",
  "usuario": "lZEJOd5tkrXEaFVHN60qdnRXrTVZApzyvUmHNp0cHc1xlQYDx/z71DMxWbq76xNiEkDcPVcf+Ko20BBTmyQ
l7WZwuo2SKbcdM9uxloxSsmBzZA7aW+Dp3Pz0yca3YUj4hd3z6uqUDn0i0hCY0/UETMXGe2WsqtZTCdKnPE2DgtNG8QbtrnxX
YEI0NZy0k4oQTf6ar9g/h35CtBpPEH/dkNlAS4HhPrgEM/JnTrgCr4K4E+ktSUKnmZ7/JIEDUTBjkdykOrs821/+zIe170Han
l2NnKLt7tX287oSU0ZqYWS4Taa+/9xiz/x/pUhREGssJj//uohHBCibZcaFJ4El1A==",
  "senha": "gOx1bqzj1OTQcDeMKq9+O3OtoqaNYVVSqKs64HzQdHVPLRT1USFLNch9ADE94GGI/0FPaOxmOIksgx5XR
+Obw1TLvx68XrPY7dZuJ3zfpBrrp2f3QRz8NmjWGbvnnyHKaKTiA9Jh47WB6nyomT/qts4h40AEBsg+k9W4+gQxBzlIeUtwhj
dp6u4u3aCFnb9h9AsGg22RJVvzKxu5twG07jy28a39WRnH2NSyC68hNoe20dWqX5uSoVpSTC5os7ba/gyHtaCZ6cbnzZrWOlB
ocwrj1vGSKNnNpLb2Aa8pyAgRTStzZwwix5byPmxAHHyOi7P67dXWLXZU+S0Ztyhq6w==",
  "scope": "read",
  "expires_in": 43199,
  "token_type": "bearer",
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzZW5oYSI6ImdPeDFicXpqMU9UUWNEZU1LcTkrT
zNPdG9xYU5ZVlZTcUtzNjRIelFkSFZQTFJUMVVTRkxOY2g5QURFOTRHR0kvMEZQYU94bU9Ja3NneDVYUitPYncxVEx2eDY4WH
JQWTdkWnVKM3pmcEJycnAyZjNRUno4Tm1qV0didm5ueUhLYUtUaUE5Smg0N1dCNm55b21UL3F0czRoNDBBRUJzZytrOVc0K2d
ReEJ6bEllVXR3aGpkcDZ1NHUzYUNGbmI5aDlBc0dnMjJSSlZ2ekt4dTV0d0cwN2p5MjhhMzlXUm5IMk5TeUM2OGhOb2UyMGRX
cVg1dVNvVnBTVEM1b3M3YmEvZ3lIdGFDWjZjYm56WnJXT2xCb2N3cmoxdkdTS05uTnBMYjJBYThweUFnUlRTdHpad3dpeDVie
VBteEFISHlPaTdQNjdkWFdMWFpVK1MwWnR5aHE2dz09IiwidXNlcl9uYW1lIjoiLTQ3NTExNDQ1MyIsInNjb3BlIjpbInJlYW
QiXSwidXN1YXJpbyI6ImxaRUpPZDV0a3JYRWFGVkhONjBxZG5SWHJUVlpBcHp5dlVtSE5wMGNIYzF4bFFZRHgvejcxRE14V2J
xNzZ4TmlFa0RjUFZjZitLbzIwQkJUbXlRbDdXWnd1bzJTS2JjZE05dXhsb3hTc21CelpBN2FXK0RwM1B6MHljYTNZVWo0aGQz
ejZ1cVVEbjBpMGhDWTAvVUVUTVhHZTJXc3F0WlRDZEtuUEUyRGd0Tkc4UWJ0cm54WFlFSTBOWnkwazRvUVRmNmFyOWcvaDM1Q
3RCcFBFSC9ka05sQVM0SGhQcmdFTS9KblRyZ0NyNEs0RStrdFNVS25tWjcvSklFRFVUQmprZHlrT3JzODIxLyt6SWUxNzBIYW
5sMk5uS0x0N3RYMjg3b1NVMFpxWVdTNFRhYSsvOXhpei94L3BVaFJFR3NzSmovL3VvaEhCQ2liWmNhRko0RWwxQT09IiwiZXh
wIjoxNDk0NjM4Njc2LCJqdGkiOiJlODMzOWM0Mi0wMzI4LTQ4ZWEtOGY4Ni1jZjczMTVjODc1NWMiLCJjbGllbnRfaWQiOiJl
eGVtcGxvIn0.YfqTTmFZ3F-FeHPapKAy_GZ2mjUkiF6aAEyD61zkgyJSfggDVCYsddQ_28MlpKco3lVvam5zglmTNRb9lAIwc
Fxc0cUdsxOyeVZXhjMx8ocHvebRlRtRCBrg3xyvQuxzRbh3lSzBw5Y6C67gvg1KtPNJEzGSHHgsdqeP3RwRmLLkfn_fOxVmj3
AWSP9NhGUSrbyMxgGGKVQOeE0lX2w72I_DlwahnPt1PP8PaYZ1E0_45gY1Idwlg5sEQt2wdN6h7LloWW4oigrVM7oMmF7_3qy
WOC1kMgqJDX1gHHQ6T_Ai-L_r_vqJyf3rorV5x7DTzvuPV0vUg_46ajMttRLqdw"
}
```


Fonte: Autoria própria (2017)

A Figura 3.38 mostra o campo access_token, obtido após o aplicativo solicitar e receber o *token*, decodificado.

Figura 3.38 – Campo access_token decodificado


```
{
    "senha": "gOx1bqzj1OTQcDeMKq9+O3OtoqaNYVVSqKs64HzQdHVPLRT1USFLNch9ADE94GGI/0FPaOxmOIksgx5XR
+Obw1TLvx68XrPY7dZuJ3zfpBrrp2f3QRz8NmjWGbvnnyHKaKTiA9Jh47WB6nyomT/qts4h40AEBsg+k9W4+gQxBzlIeUtwhj
dp6u4u3aCFnb9h9AsGg22RJVvzKxu5twG07jy28a39WRnH2NSyC68hNoe20dWqX5uSoVpSTC5os7ba/gyHtaCZ6cbnzZrWOlB
ocwrj1vGSKNnNpLb2Aa8pyAgRTStzZwwix5byPmxAHHyOi7P67dXWLXZU+S0Ztyhq6w==",
    "user_name": "-475114453",
    "scope": [
        "read"
    ],
    "usuario": "lZEJOd5tkrXEaFVHN60qdnRXrTVZApzyvUmHNp0cHc1xlQYDx/z71DMxWbq76xNiEkDcPVcf+Ko20BBTm
yQl7WZwuo2SKbcdM9uxloxSsmBzZA7aW+Dp3Pz0yca3YUj4hd3z6uqUDn0i0hCY0/UETMXGe2WsqtZTCdKnPE2DgtNG8Qbtrn
xXYEI0NZy0k4oQTf6ar9g/h35CtBpPEH/dkNlAS4HhPrgEM/JnTrgCr4K4E+ktSUKnmZ7/JIEDUTBjkdykOrs821/+zIe170H
anl2NnKLt7tX287oSU0ZqYWS4Taa+/9xiz/x/pUhREGssJj//uohHBCibZcaFJ4El1A==",
    "exp": 1494638676,
    "jti": "e8339c42-0328-48ea-8f86-cf7315c8755c",
    "client_id": "exemplo"
}
```


Fonte: Autoria própria (2017)

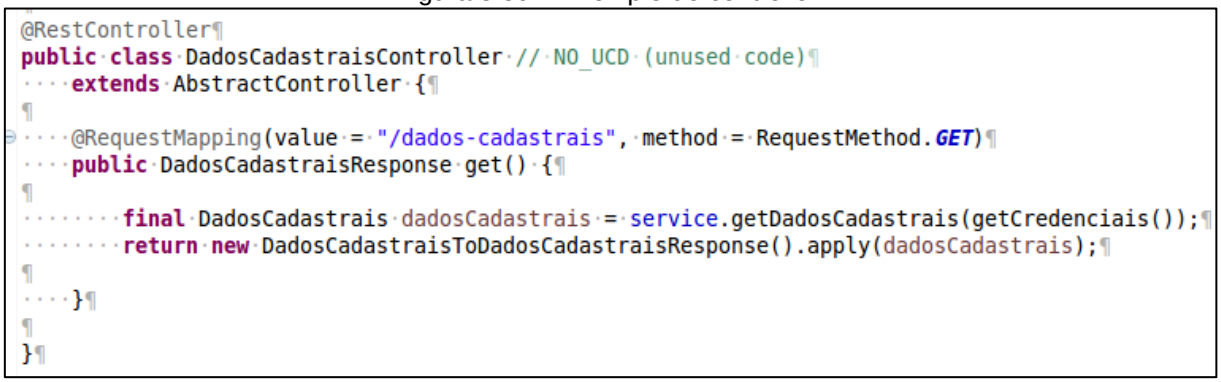
Importante ressaltar que, em nenhum momento, os dados do usuário são divulgados nas informações retornadas. Os campos usuário e senha estão criptografados e o campo user_name é um *hash* criptográfico, criados a partir das credenciais informadas. De posse do access_token, a aplicação está apta para a utilização do módulo *login*.

3.5.2 Módulo da API

O módulo API é responsável por tratar as requisições, utilizar o Conector para obter os dados do SIGA, aplicar transformações nesses dados e retorná-los ao cliente solicitante em formato JSON.

A Figura 3.39 mostra um exemplo dos *controllers* – que são as classes diretamente responsáveis pelo atendimento de cada requisição: obtendo as credenciais do usuário; executando e recebendo os dados de retorno do serviço; os transformando no formato esperado ou requisitado pelo aplicativo. A alteração para JSON é realizada automaticamente pelo *Spring Framework*, pois o *controller* é anotado com a anotação @RestController

Figura 3.39 – Exemplo de controller

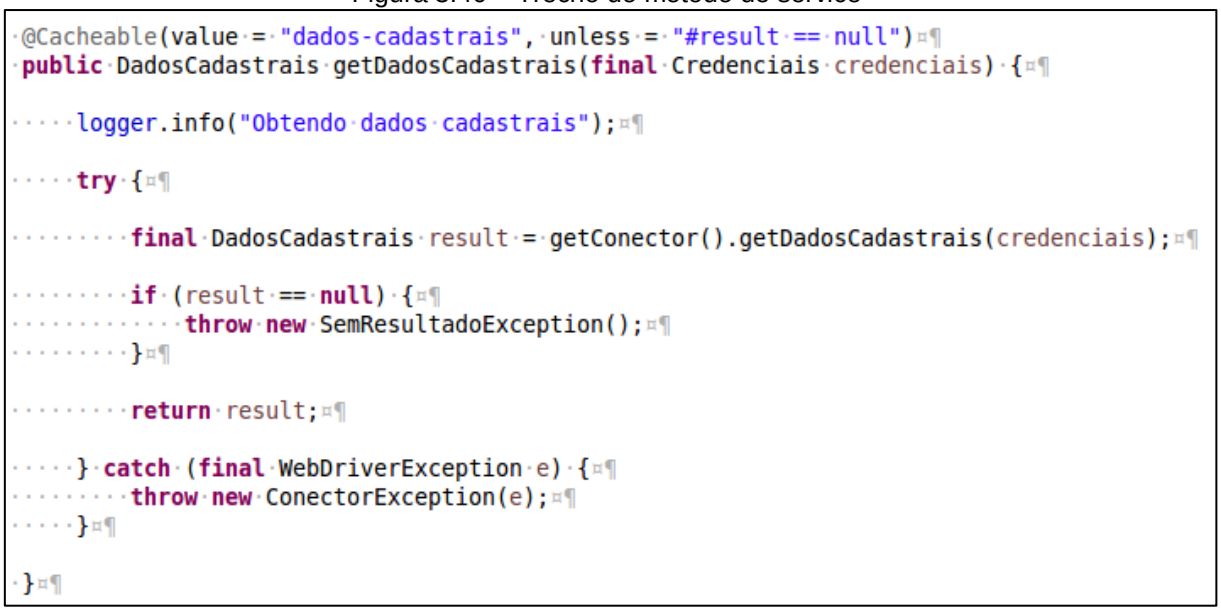


```
@RestController
public class DadosCadastraisController // NO_UCD (unused code)
    extends AbstractController {

    @RequestMapping(value = "/dados-cadastrais", method = RequestMethod.GET)
    public DadosCadastraisResponse get() {

        final DadosCadastrais dadosCadastrais = service.getDadosCadastrais(getCredenciais());
        return new DadosCadastraisToDadosCadastraisResponse().apply(dadosCadastrais);

    }

}
```

Fonte: Autoria própria (2017)

A classe de serviço principal, **ConectorService**, é responsável por realizar a chamada ao Conector e retornar, tratar eventuais erros e retornar os dados obtidos como visto no exemplo de código da Figura 3.40.

Figura 3.40 – Trecho do método de service



```
@Cacheable(value = "dados-cadastrais", unless = "#result == null")
public DadosCadastrais getDadosCadastrais(final Credenciais credenciais) {

    logger.info("Obtendo dados cadastrais");

    try {

        final DadosCadastrais result = getConector().getDadosCadastrais(credenciais);

        if (result == null) {
            throw new SemResultadoException();
        }

        return result;

    } catch (final WebDriverException e) {
        throw new ConectorException(e);
    }

}
```

Fonte: Autoria própria (2017)

A anotação @Cacheable indica que o resultado desse método deve ser armazenado em cache, desde que não seja nulo, conforme a condição exposta no atributo *unless*. Além disso, é possível notar o tratamento de exceções: caso nenhum resultado seja obtido, é lançada a exceção SemResultadoException; caso ocorra algum problema relacionado ao Selenium, indicado pela exceção WebDriverException, é lançada a exceção ConectorException.

Após a chamada ao service, o *controller* realiza o processo de transformação para retornar os dados. Essa transformação, de classes do Conector para classes locais, é uma estratégia necessária para manter a assinatura da API consistente e independente dos retornos do Conector. Isso permite que os aplicativos integrados a ela mantenham um baixo acoplamento e garantindo que nenhuma alteração no Conector gere um impacto imediato nos aplicativos dependentes da API. Esse processo é feito através de implementações da interface *Function*, padrão da API Java, sendo uma implementação para cada serviço.

Figura 3.41 – Exemplo de transformação

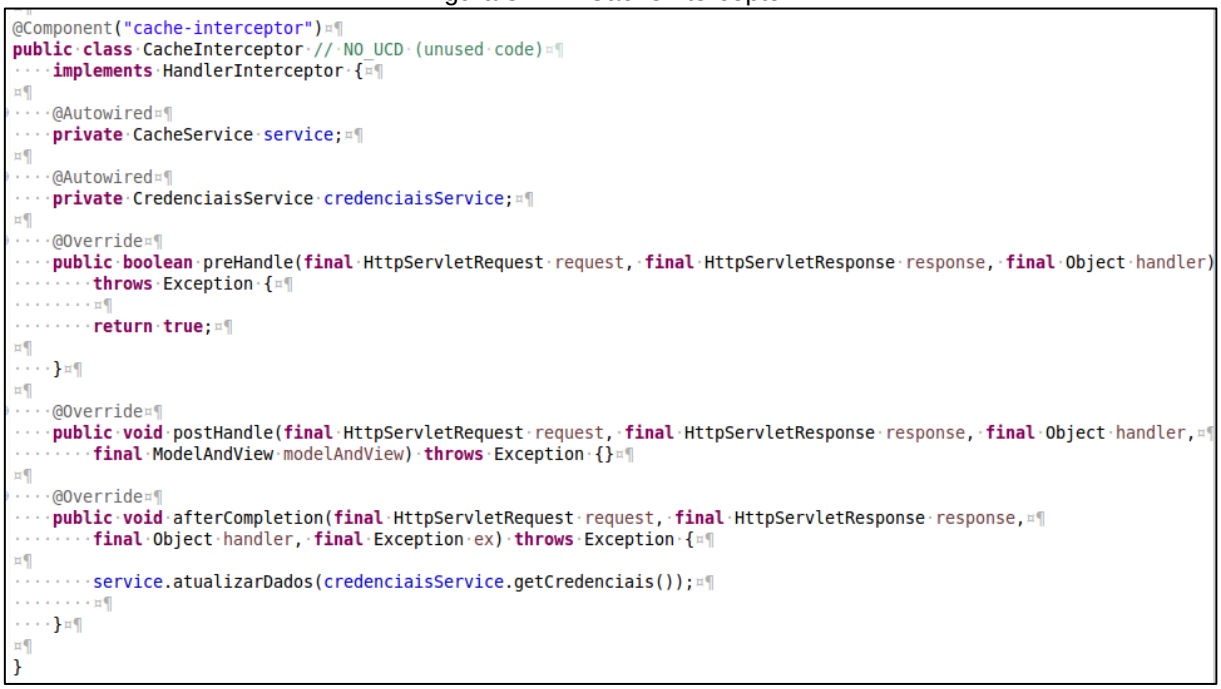
```
@Cacheable(value = "dados-cadastrais", unless = "#result == null")
public DadosCadastrais getDadosCadastrais(final Credenciais credenciais) {

    logger.info("Obtendo dados cadastrais");

    try {

        final DadosCadastrais result = getConector().getDadosCadastrais(credenciais);

        if (result == null) {
            throw new SemResultadoException();
        }

        return result;

    } catch (final WebDriverException e) {
        throw new ConectorException(e);
    }

}
```

Fonte: Autoria própria (2017)

Pode ser visto um exemplo de transformação na Figura 3.41. Um processo muito simples que consiste, basicamente, em: copiar os atributos de um objeto para o outro.

Após o retorno dos dados, é executado (de maneira assíncrona) a obtenção e armazenamento em cache de todos os dados do usuário. Essa estratégia permite que as próximas chamadas sejam executadas com um tempo menor de duração. A classe **CacheInterceptor**, na Figura 3.42, que é responsável por disparar a execução do serviço de cache após a execução de toda requisição.

Figura 3.42 – CacheInterceptor

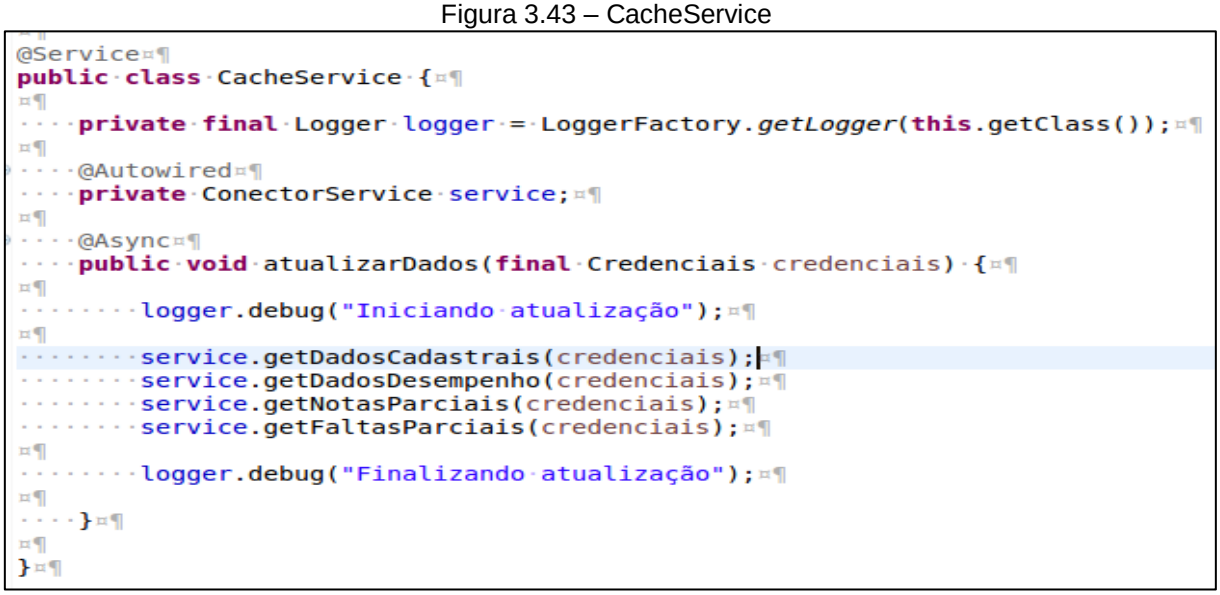


```
@Component("cache-interceptor")
public class CacheInterceptor // NO_UCD (unused code)
    implements HandlerInterceptor {

    @Autowired
    private CacheService service;

    @Autowired
    private CredenciaisService credenciaisService;

    @Override
    public boolean preHandle(final HttpServletRequest request, final HttpServletResponse response, final Object handler)
            throws Exception {

        return true;

    }

    @Override
    public void postHandle(final HttpServletRequest request, final HttpServletResponse response, final Object handler,
            final ModelAndView modelAndView) throws Exception {}

    @Override
    public void afterCompletion(final HttpServletRequest request, final HttpServletResponse response,
            final Object handler, final Exception ex) throws Exception {

        service.atualizarDados(credenciaisService.getCredenciais());

    }

}
```

Fonte: Autoria própria (2017)

O CacheService, que é apresentado na Figura 3.43 a seguir, é executado assíncronamente (devido à anotação @Async), e chama todos os serviços de obtenção de dados de maneira sequencial, fazendo com que os seus resultados sejam armazenados em cache.

Figura 3.43 – CacheService



```
@Service
public class CacheService {

    private final Logger logger = LoggerFactory.getLogger(this.getClass());

    @Autowired
    private ConectorService service;

    @Async
    public void atualizarDados(final Credenciais credenciais) {

        logger.debug("Iniciando atualização");

        service.getDadosCadastrais(credenciais);
        service.getDadosDesempenho(credenciais);
        service.getNotasParciais(credenciais);
        service.getFaltasParciais(credenciais);

        logger.debug("Finalizando atualização");

    }

}
```

Fonte: Autoria própria (2017)

Por se tratar de operações HTTP, é importante sempre retornar códigos de status que indiquem claramente o resultado da operação, seja ele de sucesso ou de erro. O Quadro 3.4 indica os códigos utilizados e seus significados.

Quadro 3.4 – Códigos e seus indicativos

Código	**Significado**
200	Sucesso
403	Acesso negado
404	Informação não encontrada
408	Tempo de operação esgotado
500	Erro inesperado
501	Método não implementado

Fonte: Autoria própria (2016)

O retorno de código de erros é controlado através do mecanismo chamado de ControllerAdvice, que consiste em alterar o retorno da chamada de acordo com a exceção capturada. Essa implementação está realizada na classe ExceptionHandlersConfig, conforme Figura 3.44.

Figura 3.44 – Trecho da classe ExceptionHandlersConfig

```
@ControllerAdvice
public class ExceptionHandlersConfig { // NO_UCD (unused code)

    private static final Logger log = LoggerFactory.getLogger(ExceptionHandlersConfig.class);

    @ExceptionHandler(SemResultadoException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    @ResponseBody
    public ErroResponse handle(final SemResultadoException e) {

        return new ErroResponse("Sem resultado");

    }

    @ExceptionHandler(NaoImplementadoException.class)
    @ResponseStatus(HttpStatus.NOT_IMPLEMENTED)
    @ResponseBody
    public ErroResponse handle(final NaoImplementadoException e) {
        return new ErroResponse("Método não implementado");
    }

    @ExceptionHandler(ConectorException.class)
    @ResponseStatus(HttpStatus.REQUEST_TIMEOUT)
    @ResponseBody
    public ErroResponse handle(final ConectorException e) {

        log.error("ConectorException", e);
        return new ErroResponse("Não foi possível conectar ao SIGA nesse momento");

    }

}
```

Fonte: Autoria própria (2017)

3.6 Testes e análise de viabilidade

Como programado, era necessário realizar testes para verificar se o projeto executava o que se pretendia e se, o mecanismo criado, reduziria a dificuldade de se desenvolver alguma ferramenta ligada ao SIGA.

3.6.1 Teste do código

No início do desenvolvimento do projeto, cada integrante executava uma classe criada para verificar se as informações trazidas eram condizentes com as exibidas no sistema.

Com a instituição dos serviços testes básicos, novas verificações foram realizadas para garantir que as informações obtidas tinham integridade. Os testes foram executados para cada requisito estabelecido.

Obtendo o sucesso nos testes básicos, a segunda etapa foi realizar a criação de testes automáticos, através da verificação de asserções, comparando dados obtidos e dados esperados. Esses testes foram realizados a cada coleta de dados locais das páginas salvas por cada integrante do projeto.

Para uma melhor visualização dos resultados dos testes automáticos, um coletor foi criado. Com isso foi possível a realização de testes com dados externos, com informações de pessoas além dos integrantes do projeto. No entanto, a conferência dos dados não foi feita pelo sistema, mas através de análise do conteúdo do HTML.

Através do coletor, foi possível testes em cenários reais, permitindo que usuários, externos ao grupo, pudessem observar o resultado da execução sem interferência dos membros deste projeto.

Devido a todos os testes realizados, e com os dados obtidos pelo coletor, foi possível acrescentar mais testes unitários aos serviços, garantindo que a aplicação funcionasse corretamente para usuário externo além dos integrantes do grupo.

Foram verificados durante os testes que, devido a lentidão do SIGA e da latência do servidor que fora escolhido para abrigar a API, o erro de número 408: tempo de operação esgotado (descrito no Quadro 3.4) pode ser apresentado.

3.6.2 Programando com a API

Depois que os códigos foram testados e a API mostrava-se funcional, era necessário que o grupo testasse a tese desenvolvida nesse projeto. Essa API iria reduzir o tempo gasto pelo desenvolvedor na criação de aplicações ligadas ao SIGA? E a resposta foi sim, o tempo gasto com o desenvolvimento caí pelo menos pela metade, se utilizando a API envolvida no projeto.

O uso de uma API reduziu drasticamente o tempo de desenvolvimento e isso pode ser verificado durante o desenvolvimento desse projeto e seu manual de uso. Desde o início, uma página PHP, cujo objetivo era exibir os dados cadastrais do usuário conectado, foi preparada. O tempo gasto foi de apenas 90 minutos. É bom ressaltar que boa parte do tempo foi consumida em preparo de ambiente de desenvolvimento, leitura da documentação da linguagem – para verificar quais funções utilizar – e atualização do manual.

Mesmo levando em consideração que o resultado ainda está longe de uma aplicação completa, é possível fazer uma projeção de que com um tempo aproximado de 16 horas seja possível obter todas as informações disponíveis na API e exibi-las em uma interface amigável.

Além disso, criou-se um cenário amplamente favorável ao desenvolvimento de novas aplicações baseadas no conjunto de dados já existentes, fazendo com que o limite seja a criatividade dos envolvidos no projeto e não mais a dificuldade de obtenção desses dados para uso sistêmico.

O projeto pode ser continuado implementando-se a obtenção dos dados que ainda não são retornados ou pode ser expandido com a criação de novos projetos que se beneficiem desse. Exemplos: um aplicativo de celular que exibe os horários da aula do dia, um sistema integra-se à agenda do usuário e insere automaticamente as datas

de provas que constam no calendário, um sistema de inscrições para eventos (como a Fatecnologia) que prioriza a inscrição de alunos de acordo com o seu curso ou instituição e até uma rede social completa criando grupos de comunicação formados automaticamente de acordo com as disciplinas em que cada um está matriculado.

3.7 Disponibilização da API

Um dos principais objetivos desse projeto é trazer utilidade à toda comunidade do Centro Paula Souza, sem qualquer tipo de restrição. Dessa forma todo e qualquer item relacionado a esse projeto estará disponível por tempo indeterminado, sob os termos da licença MIT, na organização sigapi, no GitHub, acessível pelo endereço https://github.com/sigapi

Além disso, durante o tempo de desenvolvimento, foi registrado o domínio sigapi.info para que o projeto fosse disponibilizado para testes da própria equipe e de usuários externos, sendo os principais pontos de acesso:

- API: http://api.sigapi.info
- Login: http://login.sigapi.info
- Manual de Uso: http://doc.sigapi.info

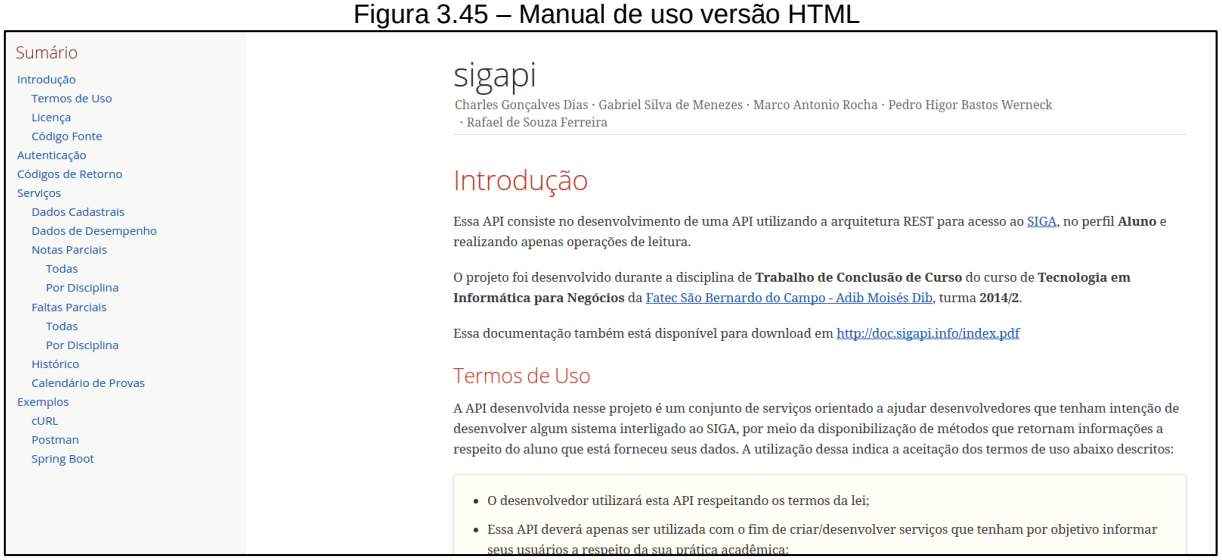
3.7.1 Manual de uso

Como qualquer sistema, o manual de uso é primordial para a correta utilização de todos os seus recursos. Por esse motivo, foi desenvolvida uma completa documentação abordando todos os pontos necessários para que qualquer desenvolvedor utilize esse projeto. Esse documento cobre desde a autenticação até exemplos de códigos.

Como todos os outros itens do projeto, a documentação foi desenvolvida utilizando-se das ferramentas padrões do *Spring Framework*, mais especificamente o *Spring* Rest Docs. Conforme o guia oficial – que pode ser encontrado em: http://docs.spring.io/spring-restdocs/docs/current/reference/html5/, é possível utilizá-lo de várias maneiras. Esse manual foi disponibilizado *online*, no caminho http://doc.sigapi.info, nos formatos HTML e PDF. O manual também foi adicionado

como apêndice nesse projeto. É possível ver na Figura 3.45 o manual de uso disponibilizado no formato HTML.

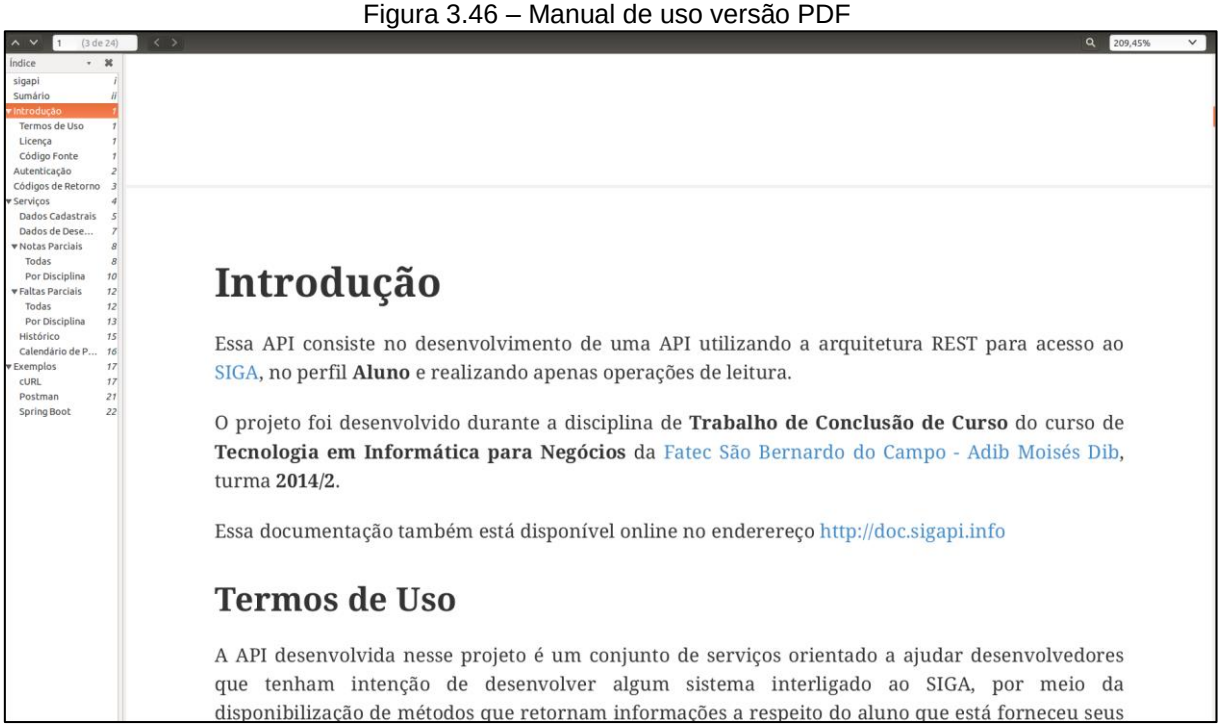
Figura 3.45 – Manual de uso versão HTML



Fonte: Autoria própria (2017)

Na Figura 3.46, pode-se verificar o manual de uso disponibilizado no formato de PDF.

Figura 3.46 – Manual de uso versão PDF



Fonte: Autoria própria (2017)

CONSIDERAÇÕES FINAIS

A vida transformou-se fortemente nas últimas décadas e não foram poucas as mudanças. A acelerada transformação tecnológica trouxe consigo dois fenômenos gêmeos: a pressa e o desenvolvimento de indivíduos multitarefas. As novas gerações provocaram, aos poucos, uma revolução tácita. Os jovens acreditam que as normas do passado não funcionam mais.

As chamadas gerações X, Y e Z – indivíduos que nasceram após os anos 80 – querem dar sentido à vida, e rápido. Tudo isso enquanto fazem outras dez coisas ao mesmo tempo. Ao passo que estudam, leem notícias na internet, checam o Facebook, escutam música e ainda prestam atenção na conversa ao lado. Para essa geração, a velocidade da notícia e do mundo é outra. Os resultados precisam ser quase que instantâneos, e os desafios, constantes.

Com toda essa mudança, é preciso, antes de tudo, aprender a conversar com essa turma para revelar seu potencial. E essa conversa pode ser ao vivo, por e-mail, pelo Messenger, pelo WhatsApp, celular, MSN, Twitter ou qualquer outra ferramenta que venha a surgir no mundo. Uma pesquisa realizada pelo Departamento de Educação dos Estados Unidos demonstrou[1]: crianças que usam programas *online* para aprender ficam nove pontos acima da média geral dos demais estudantes e são mais motivadas.

Pensando nessa mudança comportamental, esse projetou criou estruturas para ampliar a integração digital da FATEC e fabricou meios para fomentar o desenvolvimento de aplicações que possam ir ao encontro da velocidade das novas gerações. Esse projeto tenta conversar de forma adequada com a posteridade que, desde o berço, estará inserida na tecnologia. O projeto trouxe o conteúdo de novas tecnologias para o grupo e para a comunidade acadêmica, na tentativa de inserir o SIGA no que, hoje, é praticamente comum nos jovens: a utilização de aplicativos em smartphones.

1. Pesquisa noticiada pela revista Galileu (Loiola, 2009).

A estrutura alcançou seu objetivo. Foi possível verificar que o desenvolvimento de uma aplicação qualquer, utilizando o conteúdo disponibilizado no SIGA, se tornou mais fácil com o suporte da API. Corroborou-se que é muito importante ter desenvolvedores criando estruturas para facilitar a criação de novas aplicações. Assim como, em algum momento, o Eclipse foi desenvolvido para facilitar a construção de aplicações Java, será diferencial ter ferramentas que facilitem o aperfeiçoamento do SIGA. Esses alicerces fortalecem toda uma comunidade de desenvolvedores e impulsionam a criação de novas aplicações.

O Facebook, Youtube, LinkedIn, e as demais redes sociais, hoje, são fortalecidas com suas integrações. Qualquer desenvolvedor, atualmente, procura integrar sua aplicação a uma dessas redes. A API é o motivo para, sem ter que criar um cadastro, as pessoas realizarem o acesso à uma página no menor tempo possível. Por causa de uma API, as informações são rapidamente compartilhadas de uma rede para outra e evitar de preencher formulários extensos. Saber que as novas gerações necessitam de velocidade foi o ponto chave dessas empresas. Seguindo esse caminho, esse projeto apresentou um meio para que o SIGA passe a fazer parte das aplicações integradas.

Para a comunidade ficam as possibilidades de sistemas, aplicações e utilitários. Aos próximos alunos, uma porta, para obter as informações do sistema de gestão acadêmica, foi aberta. Que esse projeto possa impulsionar as pessoas em atravessá-la. Que a tecnologia, aqui demonstrada, possa promover a descoberta de novas ferramentas. Para a comunidade fica mais um tijolo para o enorme muro de conhecimento que, construído em conjunto, fará com que a humanidade chegue cada vez mais longe.

REFERÊNCIAS

ALECRIM, Emerson. **Criptografia**, 2005. Disponível em: <http://www.infowester.com/criptografia.php>. Acesso em: 24 out. 2016.

AFONSO, A. et al. **Manual de Tecnologias da Informação e Comunicação e OpenOffice.org**. 2.ed. Lisboa: ANJAF, 2010.

BACHIEGA, Leandro C. **Aplicações de Tecnologia Java no Desenvolvimento de Portais para Redes Sociais e Inclusão Digital**. Graduado, Instituto Municipal de Ensino Superior de Assis, 2008

BALLEM, Marcio. **Capturando conteúdo HTML com Jsoup**, 2012. Disponível em: <http://www.mballem.com/post/capturando-conteudo-html-com-jsoup/>. Acesso em: 16 out. 2016.

CASTRO, J. L. A.; MATOS, R. L. S. **Protocolo de coerência de memória cache para sistemas distribuídos**. Revista Técnica de la Facultad de Ingeniería Universidad del Zulia, v. 30, n. 2, p. 170-178, 2007. Disponível em: <http://www.scielo.org.ve/scielo.php?script=sci_arttext&pid=S0254-07702007000200008&lang=pt>. Acesso em: 24 out. 2016.

CENTRO PAULA SOUZA. Perfil Histórico. <http://www.cps.sp.gov.br/quem-somos/perfil-historico/> Acesso em: 07 dez. 2016.

CERVO, A. L.; BERVIAN, P. A.; SILVA, R. **Metodologia científica**. 6ª. ed. São Paulo: Prentice Hall, 2007.

CHUVAKIN, A. et al. **Logging and log management**. Waltham, Mass.: Syngress, 2013.

CLARO, D. B.; SOBRAL, J. B. M. **Programação em JAVA** Florianópolis, SC: Pearson Education, 2008.

COMPARATO, Fábio K. **Ética: direito, moral e religião no mundo moderno**. São Paulo: Companhia das letras, 2006.

COMPUTAÇÃO UFCG. **Frameworks**. Disponível em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/oque.htm>. Acesso em: 15 out. 2016.

CRUZ, Edilson F. da. **A Criptografia e seu Papel na Segurança da Informação e das Comunicações (SIC)**. Especialização, Universidade de Brasília, 2009.

DEVMEDIA. **Conheça o Apache Tomcat.** Disponível em: <http://www.devmedia.com.br/conheca-o-apache-tomcat/4546>. Acesso em: 5 out. 2016.

DIAS, André. **Vantagens e Desvantagens do Controle de Versão Distribuído**, 2016. Disponível em: <https://blog.pronus.io/posts/vantagens-e-desvantagens-do-controle-de-versao-distribuido/>. Acesso em: 1 out. 2016.

DOPHINE, Thiago M. **OAuth2 - uma abordagem para segurança de aplicações e APIs REST**, 2014. Disponível em: <https://www.infoq.com/br/presentations/oauth2-uma-bordagem-para-seguranca>. Acesso em: 9 out. 2016.

ECMA INTERNATIONAL. **Standard ECMA-404** 1.ed. p. 2: 2013. Disponível em: < http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf>. Acesso em: 4 out. 2016.

FARIA, F. B. et al. **Evolução e Principais Características do IDE Eclipse**, 2010. Disponível em: <http://www.enacomp.com.br/2010/cd/artigos/completos/enacomp2010_23.pdf>. Acesso em: 23 out. 2016.

FEMENICK, Tomislav R. **Conceitos Fundamentais Sobre custos**, 2005. Disponível em: <http://www.tomislav.com.br/conceitos-funtamentais-sobre-custos/ >. Acesso: em 02 nov. 2016.

FOWLER, Martin. **Inversion Of Control**, 2005. Disponível em: < http://martinfowler.com/bliki/InversionOfControl.html>. Acesso em: 26 out. 2016.

FUNDAÇÃO GETÚLIO VARGAS. Tecnologia da Informação: Pesquisa Anual do Uso de TI. 27ª. São Paulo: FGV-EAESP, 2016.

_____. **Inversion of Control Containers and the Dependency Injection pattern**, 2004. Disponível em: <http://www.martinfowler.com/articles/injection.html>. Acesso em: 16 out. 2016.

FREITAS, Daniel T. M. de. **Análise Comparativa entre Sistemas de Controle de Versões**. Bacharelado, Universidade Federal de Juiz de Fora, 2010.

GAMA, Alexandre. **Inversão de Controle x Injeção de Dependência**. Disponível em: < http://www.devmedia.com.br/inversao-de-controle-x-injecao-de-dependencia/18763 >. Acesso em: 26 out. 2016.

GARRETT, Filipe. **O que é criptografia?**, 2012. Disponível em: <http://www.techtudo.com.br/artigos/noticia/2012/06/o-que-e-criptografia.html>. Acesso em: 9 out. 2016.

GIL, Antonio C. **Como Elaborar Projetos de Pesquisa**. São Paulo: Atlas, 2002. 176p.

GOMES, Fabio. **TDD**: fundamentos do desenvolvimento orientado a testes, 2016. <http://www.devmedia.com.br/tdd-fundamentos-do-desenvolvimento-orientado-a-testes/28151> Acesso em: 24 nov. 2016.

GUPTA, Samudra. **Pro Apache log4j**. Berkeley, Calif.: Apress, 2005.

HEATON, Jeff. **Programming Spiders, Bots, and Aggregators in Java™**. 1 ed. CA: Sybex, 2002.

HOUSTON, Pete. **Instant Jsoup How-to**. Birmingham: Packt Publishing, 2013.

JACOB, Bruce, NG, Spencer, WANG, David T. **Memory systems**. Burlington, MA: Morgan Kaufmann Publishers, 2008.

JACOBSON, D.; BRAIL, G.; WOODS, D. **APIs - da strategy guide**. Beijing: O'Reilly, 2011.

JOHNSON, R. E; FOOTE, B. **Designing Reusable Classes**, disponível em: <http://www.laputan.org/drc.html>. Acesso em: 15 out. 2016.

JSON WEB TOKENS. **Introduction to JSON Web Tokens**. Disponível em: <https://jwt.io/introduction/>. Acesso em: 27 out. 2016.

LAKATOS, E. M.; MARCONI, M. de A. **Fundamentos de metodologia científica**. 6. ed. 5. Reimp. São Paulo: Atlas, 2007.

LOIOLA, RITA. Geração Y. Galileu, 219. São Paulo: Editora Globo S.A., ano 23, n.1897, p. 23-24, out. 2009.

MACÊDO, Diego. **Gerenciamento dos Riscos do Projeto:** PMBoK 5ª ed, 2014. Disponível em:< http://www.diegomacedo.com.br/gerenciamento-dos-riscos-do-projeto-pmbok-5a-ed/ >. Acesso em 02 nov. 2016.

MASIERO, Paulo Cesar. **Ética em Computação**. São Paulo: Ed. da Universidade de São Paulo, 2000. 216 p.

MEDEIROS, Higor. **Introduzindo o servidor de aplicação Apache Tomcat**. Disponível em: <http://www.devmedia.com.br/introduzindo-o-servidor-de-aplicacao-apache-tomcat/27939>. Acesso em: 13 out. 2016.

_____. **Mocks**: Introdução a Automatização de Testes com Mock Object, 2016 <http://www.devmedia.com.br/mocks-introducao-a-automatizacao-de-testes-com-mock-object/30641> Acesso em: 24 nov. 2016.

MICELI, Rafael. **JWT (JSON Web Token**). Disponível em: <http://rafael-miceli.com.br/oauth/2016/04/18/JWT.html>. Acesso em: 8 out. 2016.

MITCHELL, Ryan. **Web scraping with python:** Collecting Data from the Modern Web. 1 ed. CA: O'Reilly, 2015.

MULLOY, Brian. **Web API design**: Crafting Interfaces that Developers Love. 1 ed. [S.L.]: Apigee, 2012

NENE, Dhananjay A. **Beginners guide to Dependency Injection**, 2005. Disponível em: <http://www.theserverside.com/news/1321158/A-beginners-guide-to-Dependency-Injection>. Acesso em: 16 out. 2016.

NIJIM, S.; PAGANO, B. **API for dummies**: A wiley brand. Hoboken, NJ: Apigee, 2014.

NOVAES, Rafael. **O que é e para que serve IDE?**, 2014. Disponível em: <http://www.psafe.com/blog/o-que-serve-ide/>. Acesso em: 16 out. 2016.

OPEN SOURCE INITIATIVE. **The MIT license (MIT)**. Disponível em: <https://opensource.org/licenses/mit>. Acesso em: 22 nov. 2016.

ORRICO, Hugo Jr. **Pirataria de Software**. 2.ed. São Paulo: MM Livros, 2004.

OTTERO, Rodrigo. **Introdução ao Maven**. Disponível em: <http://www.devmedia.com.br/introducao-ao-maven/25128>. Acesso em: 9 out. 2016.

PERNAMBUCO, I. A. M. Jr.; SANTOS, G. R.; VALENTE, W. A. G. **Sistemas de controle de versão:** Revista Engenharia de Software Magazine 49. Disponível em: < http://www.devmedia.com.br/sistemas-de-controle-de-versao-revista-engenharia-de-software-magazine-49/24745>. Acesso em: 22 de out. 2016.

PRESSMAN, Roger S. **Engenharia de software**: Uma Abordagem Profissional. 7 ed. São Paulo: Amgh, 2011. 771 p.

PRODANOV, C. C.; FREITAS, E. C. **Metodologia do trabalho científico:** Métodos e Técnicas da Pesquisa e do Trabalho Acadêmico. 2. ed. Rio Grande do Sul: Feevale, 2013. 277 p.

PROGRAM CREEK. **What is Servlet Container ?**, 2013. Disponível em: <http://www.programcreek.com/2013/04/what-is-servlet-container/>. Acesso em: 24 out. 2016.

REGISTRO BR. Disponível em: <http://registro.br/sobre/>. Acesso em: 02 nov. 2016.

SAMPAIO, Cleuton. **Web 2.0 e Mashups**: Reinventando a Internet. 1.ed. Rio de Janeiro: Brasport 2007.

SAMS, Prashanth. **Selenium Essentials**. Birmingham: Packt Publishing, 2015.

SIRIWARDENA, Prabath. **Advanced API security**. New York: Apress, 2014.

SOMMERVILLE, Ian. **Engenharia de Software: 8ed**. Rio de Janeiro: Prentice-Hall, 2008.

SPASOVSKI, Martin. **OAuth 2.0 identity and access management patterns**. Birmingham: Packt, 2013.

TOFFOLO, Túlio. **Controle de Versão com GIT**, 2011. Disponível em: <http://www.decom.ufop.br/toffolo/site_media/uploads/2011-1/git.pdf>. Acesso em: 16 out. 2016.

TRIBUNAL DE CONTAS DA UNIÃO. **Boas práticas em segurança da informação**. 2ed. Brasília, 2007. 70 p.
UOL HOST. Disponível em: <http://www.uolhost.uol.com.br/hospedagem-de-sites/linguagem-programacao/java.html#rmcl>. Acesso em: 02 nov. 2016.

VENTURA, Plinio. **O que é Requisito Funcional,** 2016.
Disponível em: <http://www.ateomomento.com.br/o-que-e-requisito-funcional/>
Acesso em: 23 mar. 2017.

WEBSTER, Susanne. **What Is Scraping? The Basics For Everyone**, 2005.
Disponível em: <https://myhelpster.com/what-is-scraping-the-basics-for-everyone/>.
Acesso em: 31 out. 2016.

APÊNDICES

APÊNDICE A – TEXTO DA LICENÇA DA API

The MIT License (MIT)

Copyright 2016: Charles Gonçalves Dias, Gabriel Silva de Menezes, Marco Antonio Rocha, Pedro Higor Bastos Werneck and Rafael de Souza Ferreira

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

APÊNDICE B – MANUAL DE UTILIZAÇÃO DA API

sigapi

Charles Gonçalves Dias, Gabriel Silva de Menezes, Marco Antonio Rocha, Pedro Higor Bastos Werneck, Rafael de Souza Ferreira

Sumário

Introdução

Essa API consiste no desenvolvimento de uma API utilizando a arquitetura REST para acesso ao SIGA, no perfil **Aluno** e realizando apenas operações de leitura.

O projeto foi desenvolvido durante a disciplina de **Trabalho de Conclusão de Curso** do curso de **Tecnologia em Informática para Negócios** da Fatec São Bernardo do Campo - Adib Moisés Dib, turma **2014/2**.

Essa documentação também está disponível online no endereço http://doc.sigapi.info

Termos de Uso

A API desenvolvida nesse projeto é um conjunto de serviços orientado a ajudar desenvolvedores que tenham intenção de desenvolver algum sistema interligado ao SIGA, por meio da disponibilização de métodos que retornam informações a respeito do aluno que está forneceu seus dados. A utilização dessa indica a aceitação dos termos de uso abaixo descritos:

- O desenvolvedor utilizará esta API respeitando os termos da lei;
- Essa API deverá apenas ser utilizada com o fim de criar/desenvolver serviços que tenham por objetivo informar seus usuários a respeito da sua prática acadêmica;
- Essa API não garante a veracidade e corretude dos dados, já que ela apenas retorna o que está sendo exibido no SIGA, estando assim vulnerável a seus erros e problemas. Assim sendo, a equipe do projeto não se responsabiliza por falhas ou inexatidões que possam estar sendo exibidas.

Licença

Projeto licenciado sob a licença MIT.

Código Fonte

O código fonte está disponível no GitHub

Autenticação

A autenticação dessa API é realizada externamente, através do servidor OAuth2 desenvolvido especialmente para esse projeto, e utilizando-se do fluxo *Authorization Code* e um Token JWT.

Por padrão, é disponibilizado um conjunto de dados para que o interessado realize o desenvolvimento da aplicação.

Dados de acesso

- **Client ID**: exemplo
- **Client Secret**: exemplo
- **Auth URL**: http://login.sigapi.info/oauth/authorize?response_type=code&redirect_uri=http://localhost
- **Access Token URL**: http://login.sigapi.info/oauth/token?redirect_uri=http://localhost
- **Tempo de validade do Token**: 30 minutos

Lembre-se de substituir o parâmetro **redirect_uri** pela URI da sua aplicação

Os dados de acesso aqui disponíveis devem ser utilizados apenas em ambiente de desenvolvimento.

Para utilização em uma aplicação em ambiente de produção, crie um ticket no GitHub solicitando as suas credenciais

Cada linguagem ou ferramenta possui métodos para tratar o fluxo de obtenção do Token. Caso necessário, verifique nossos exemplos.

Códigos de Retorno

Código HTTP	Descrição
200	OK
403	Acesso negado
404	Informação não encontrada
408	Tempo de operação esgotado
500	Erro inesperado
501	Método não implementado

Serviços

Os dados retornados pelos são armazenados temporariamente em um cache local. Nenhum dado do aluno é salvo em nenhum momento.

Todas as chamadas são autenticadas, dessa forma o Token de autenticação deve **sempre** ser informado.

Dados Cadastrais

Dados cadastrais, obtidos do perfil do aluno.

Requisição cURL

```
$ curl 'http://api.sigapi.info/api/dados-cadastrais' -i \
    -H 'Authorization: Bearer ${TOKEN}'
```

Requisição HTTP

```
GET /api/dados-cadastrais HTTP/1.1
Authorization: Bearer ${TOKEN}
Host: api.sigapi.info
```

Resposta

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "nome" : "Juca da Silva",
  "foto" :
"https://sigacentropaulasouza.com.br/imagem/Z33YVAVR4NXTBKX8UJL6X9KMN35SR5.TMB.JPG",
  "ra" : "98657842147",
  "instituicao" : "Faculdade de Tecnologia de São Bernardo do Campo \"Adib Moises
Dib\"",
  "curso" : "Tecnologia em Informática para Negócios",
  "turno" : "Noturno",
  "emailFatec" : "nome.sobrenome@fatec.sp.gov.br",
  "emailWebsai" : "nome.sobrenome@gmail.com",
  "emailPreferencial" : "nome.sobrenome@gmail.com",
  "outrosEmails" : [ "nome.sobrenome@outlook.com", "nome.sobrenome@yahoo.com.br" ]
}
```

Descrição da Resposta

Path	Type	Description
nome	String	Nome do aluno
foto	String	URL da foto do aluno
ra	String	Registro acadêmico do aluno
instituicao	String	Instituição matriculada
curso	String	Curso matriculado
turno	String	Turno matriculado
emailEtec	String	Email cadastrado na ETECT
emailFatec	String	Email cadastrado na FATEC

Path	Type	Description
emailWebsai	String	Email cadastrado no WEBSAI
emailPreferencial	String	Email cadastrado como preferencial
outrosEmails	Array	Outros emails cadastrados

Dados de Desempenho

Dados de desempenho, obtidos do perfil do aluno.

Requisição cURL

```
$ curl 'http://api.sigapi.info/api/dados-desempenho' -i \
    -H 'Authorization: Bearer ${TOKEN}'
```

Requisição HTTP

```
GET /api/dados-desempenho HTTP/1.1
Authorization: Bearer ${TOKEN}
Host: api.sigapi.info
```

Resposta

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "pp" : 76.66,
  "pr" : 8.35,
  "maiorPrCurso" : 9.3
}
```

Descrição da Resposta

Path	Type	Description
pp	Number	Índice PP do aluno
pr	Number	Índice PR do aluno
maiorPrCurso	Number	Maior PR do curso

Notas Parciais

Dados obtidos através do link de **Notas Parciais**.

Todas

Todas as notas parciais.

Requisição cURL

```
$ curl 'http://api.sigapi.info/api/notas-parciais' -i \
    -H 'Authorization: Bearer ${TOKEN}'
```

Requisição HTTP

```
GET /api/notas-parciais HTTP/1.1
Authorization: Bearer ${TOKEN}
Host: api.sigapi.info
```

Resposta

```
HTTP/1.1 200 OK
Content-Type: application/json

[ {
  "siglaDisciplina" : "AGO007",
  "nomeDisciplina" : "Gestão e Operação por Processos",
  "avaliacoes" : {
    "Avaliação Oficial - P3" : 0,
    "Trabalho" : 0,
    "Avaliação Oficial - P1" : 9,
    "Avaliação Oficial - P2" : 0
  },
  "mediaFinal" : 2.8,
  "quantidadeFaltas" : 0,
  "percentualFrequencia" : 0
}, {
  "siglaDisciplina" : "IMH002",
  "nomeDisciplina" : "Multimídia e Hipermídia",
  "avaliacoes" : {
    "Avaliação Oficial - P3" : 0,
    "Trabalho" : 10,
    "Avaliação Oficial - P1" : 8.5,
    "Avaliação Oficial - P2" : 0
  },
  "mediaFinal" : 4.7,
  "quantidadeFaltas" : 0,
  "percentualFrequencia" : 0
}, {
```

```
  "siglaDisciplina" : "ISA001",
  "nomeDisciplina" : "Fundamentos de Auditoria",
  "avaliacoes" : {
    "Avaliação Oficial - P3" : 0,
    "Trabalho" : 0,
    "Avaliação Oficial - P1" : 8,
    "Avaliação Oficial - P2" : 0
  },
  "mediaFinal" : 2.5,
  "quantidadeFaltas" : 0,
  "percentualFrequencia" : 0
}, {
  "siglaDisciplina" : "ISJ002",
  "nomeDisciplina" : "Sistemas de Gestão de Produção e Logística",
  "avaliacoes" : {
    "Trabalho" : 0,
    "Avaliação Oficial - P3" : 0,
    "Avaliação Oficial - P1" : 8,
    "Avaliação Oficial - P2" : 0
  },
  "mediaFinal" : 2.5,
  "quantidadeFaltas" : 8,
  "percentualFrequencia" : 90
}, {
  "siglaDisciplina" : "ITI104",
  "nomeDisciplina" : "Governança em Tecnologia da Informação",
  "avaliacoes" : {
    "Avaliação Oficial - P3" : 0,
    "Trabalho" : 0,
    "Avaliação Oficial - P1" : 8.5,
    "Avaliação Oficial - P2" : 0
  },
  "mediaFinal" : 2.7,
  "quantidadeFaltas" : 0,
  "percentualFrequencia" : 0
}, {
  "siglaDisciplina" : "LIN600",
  "nomeDisciplina" : "Inglês VI",
  "avaliacoes" : { },
  "mediaFinal" : 0,
  "quantidadeFaltas" : 0,
  "percentualFrequencia" : 0
}, {
  "siglaDisciplina" : "TES001",
  "nomeDisciplina" : "Estágio Supervisionado",
  "avaliacoes" : {
    "Relatório de Estágio Supervisionado" : 0
  },
  "mediaFinal" : 3,
  "quantidadeFaltas" : 0,
  "percentualFrequencia" : 0
```

```
}, {
  "siglaDisciplina" : "TTG103",
  "nomeDisciplina" : "Trabalho de Graduação II",
  "avaliacoes" : { },
  "mediaFinal" : 0,
  "quantidadeFaltas" : 0,
  "percentualFrequencia" : 0
}, {
  "siglaDisciplina" : "TTG202",
  "nomeDisciplina" : "Projeto de Trabalho de Graduação",
  "avaliacoes" : { },
  "mediaFinal" : 0,
  "quantidadeFaltas" : 0,
  "percentualFrequencia" : 0
} ]
```

Descrição da Resposta

Path	Type	Description
`[]`	`Array`	Listagem de notas parciais
`[].mediaFinal`	`Number`	Média final
`[].quantidadeFaltas`	`Number`	Quantidade de faltas
`[].percentualFrequencia`	`Number`	Percentual de frequência
`[].avaliacoes`	`Object`	Avaliações
`[].avaliacoes.*`	`Number`	Nota por avaliação
`[].siglaDisciplina`	`String`	Sigla da disciplina
`[].nomeDisciplina`	`String`	Nome da disciplina

Por Disciplina

Obtenção das notas parciais de determinada disciplina.

/api/notas-parciais/{sigla}

Parameter	Description
`sigla`	Sigla da disciplina desejada

Obtenção das notas parciais de determinada disciplina.

Requisição cURL

```
$ curl 'http://api.sigapi.info/api/notas-parciais/AGO007' -i \
    -H 'Authorization: Bearer ${TOKEN}'
```

Requisição HTTP

```
GET /api/notas-parciais/AGO007 HTTP/1.1
Authorization: Bearer ${TOKEN}
Host: api.sigapi.info
```

Resposta

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "siglaDisciplina" : "AGO007",
  "nomeDisciplina" : "Gestão e Operação por Processos",
  "avaliacoes" : {
    "Avaliação Oficial - P3" : 0,
    "Trabalho" : 0,
    "Avaliação Oficial - P1" : 9,
    "Avaliação Oficial - P2" : 0
  },
  "mediaFinal" : 2.8,
  "quantidadeFaltas" : 0,
  "percentualFrequencia" : 0
}
```

Descrição da Resposta

Path	Type	Description
mediaFinal	Number	Média final
quantidadeFaltas	Number	Quantidade de faltas
percentualFrequencia	Number	Percentual de frequência
avaliacoes	Object	Avaliações
avaliacoes.*	Array	Nota por avaliação
siglaDisciplina	String	Sigla da disciplina
nomeDisciplina	String	Nome da disciplina

Faltas Parciais

Dados obtidos através do link **Faltas Parciais**.

Todas

Todas as faltas parciais.

Requisição cURL

```
$ curl 'http://api.sigapi.info/api/faltas-parciais' -i \
    -H 'Authorization: Bearer ${TOKEN}'
```

Requisição HTTP

```
GET /api/faltas-parciais HTTP/1.1
Authorization: Bearer ${TOKEN}
Host: api.sigapi.info
```

Resposta

```
HTTP/1.1 200 OK
Content-Type: application/json

[ {
  "siglaDisciplina" : "IMH002",
  "nomeDisciplina" : "Multimídia e Hipermídia",
  "quantidadeAusencias" : 16,
  "quantidadePresencas" : 36
}, {
  "siglaDisciplina" : "ISA001",
  "nomeDisciplina" : "Fundamentos de Auditoria",
  "quantidadeAusencias" : 10,
  "quantidadePresencas" : 14
}, {
  "siglaDisciplina" : "ITI104",
  "nomeDisciplina" : "Governança em Tecnologia da Informação",
  "quantidadeAusencias" : 5,
  "quantidadePresencas" : 21
}, {
  "siglaDisciplina" : "TES001",
  "nomeDisciplina" : "Estágio Supervisionado",
  "quantidadeAusencias" : 0,
  "quantidadePresencas" : 0
}, {
  "siglaDisciplina" : "TTG202",
  "nomeDisciplina" : "Projeto de Trabalho de Graduação",
  "quantidadeAusencias" : 0,
  "quantidadePresencas" : 18
} ]
```

Descrição da Resposta

Path	Type	Description
`[]`	`Array`	Listagem de faltas parciais
`[].quantidadeAusencias`	`Number`	Quantidade de ausências
`[].quantidadePresencas`	`Number`	Quantidade de presenças
`[].siglaDisciplina`	`String`	Sigla da disciplina
`[].nomeDisciplina`	`String`	Nome da disciplina

Por Disciplina

Obtenção das faltas parciais de determinada disciplina.

/api/faltas-parciais/{sigla}

Parameter	Description
`sigla`	Sigla da disciplina desejada

Obtenção das faltas parciais de determinada disciplina.

Requisição cURL

```
$ curl 'http://api.sigapi.info/api/faltas-parciais/IMH002' -i \
    -H 'Authorization: Bearer ${TOKEN}'
```

Requisição HTTP

```
GET /api/faltas-parciais/IMH002 HTTP/1.1
Authorization: Bearer ${TOKEN}
Host: api.sigapi.info
```

Resposta

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "siglaDisciplina" : "IMH002",
  "nomeDisciplina" : "Multimídia e Hipermídia",
  "quantidadeAusencias" : 16,
  "quantidadePresencas" : 36
}
```

Descrição da Resposta

Path	Type	Description
quantidadeAusencias	Number	Quantidade de ausências
quantidadePresencas	Number	Quantidade de presenças
siglaDisciplina	String	Sigla da disciplina
nomeDisciplina	String	Nome da disciplina

Histórico

Dados de histórico de conclusão do curso, obtidos através do link **Histórico Completo**.

Serviço ainda não implementado

Calendário de Provas

Dados do calendário de provas, obtidos através do link **Calendário de Provas**.

Serviço ainda não implementado

Exemplos

Abaixo estão alguns exemplos de uso da API.

A equipe do projeto é responsável pela manutenção e suporte da API e cada desenvolvedor é responsável pela integração à sua aplicação. Os exemplos abaixo devem ser usados apenas como um ponto de partida.

cURL

A obtenção do código de autorização deve ser feita diretamente pelo navegador

- Obtenha o código de autorização
  a. Acesse o endereço http://login.sigapi.info/oauth/authorize?client_id=exemplo&redirect_uri=http://localhost&response_type=code
  b. Na tela que se abre, preencha seu usuário e senha e clique no botão **login**
  c. Na tela de autorização, clique no botão **Authorize**
  d. O browser será redirecionado para o *localhost* e, ao final do endereço, será exibido o código de autorização
     - http://localhost/?code=**N5JB9h**
  e. Salve o código de autorização obtido
- Obtenha o token de acesso
  a. Faça a requisição para obter o Token

```
$ client_id=exemplo
$ client_secret=exemplo
$ authorization_code=N5JB9h
$ curl -u ${client_id}:${client_secret} -X POST
"http://login.sigapi.info/oauth/token?code=${authorization_code}&grant_type=auth
orization_code&redirect_uri=http://localhost"
```

  b. Verifique o retorno da requisição

```
{
  "jti": "8c2748c6321-39a2-42f6-b68d-709355676fde",
  "usuario":
"L123Q9Ll6UqaIHDX9bNadjYc9NRdvzCtqFHCz4zXmWPnmHl+BpXXTHBvPynBLSs1dyWkl53bHaGN7LL
S6eWhTxqMTXIqh7HMPTmc53fzNJ9Gymj9pSZ0mFuCyFa23bbdwoZJag7b3xupeAQCVJbv9VQaGzQyc29
3DGrPVCV1DyS6mtfJzmzUrEv+B4+bAcbFi1JgctDEpFoCSOKd1sh28DjIBS+OKyq+4LrIu27HE8D9Ktw
MZZqymajg8Y0vpY3qrrO0tHmpZs+hW9k/PXROS6txGNYtE4/3lY1B6xmCn8F9gLHYLSB6irQ+G2YBPSb
6Et2wvo0hWH61R+yrs4DV8QPpA==",
  "senha":
"HtWBEVavdmTv123y/pdXm1fjWYzD47ljEdEYJl8n9VylUFR3/gaI8VAv6QVuHkGpEBiQIvb61k0gGAn
/7X7iq787O3BU8k2Cie8fMJXIJV23j+l55WgI6hY85RiFSzN4GFb3NRIUpMFubdWkR7/q05rIXov9w+H
4bx2YPfr71DqQ16puwHfZM5BEVMJz8FXGH/SiVInRaPQv/a1zwNmDygBF9jSETgSMzjosh+7ah3UKJVy
PGs/t46CeIh9/3SwyqBRtV38U+lg4sOvT+jI20rExe3xnur4s+8jKzM9pZGOjRfdv70s+hC3Mrng3NOm
DEu1AFBpd/vw0Nzijz8XZfKyMA==",
  "scope": "read",
  "expires_in": 43199,
  "token_type": "bearer",
  "access_token":
"eyJhbGciOiJSUzI1Ni123IsInR5cCI6IkpXVCJ9.eyJzZW5oYSI6Ikh0V0JFVmF2ZG1UdnkvcGRYbTF
maldZekQ0N2xqRWRFWUpsOG45VnlsVUZSMy9nYUk4VkF2NlFWdUhrR3BFQmlRSXZiNjFrMGdHQW4vN1g
3aXE3ODdPM0JVOGsyQ2llOGZNSlhJSlYyM2orbDU1V2dJNmhZODVSaUZTek40R0ZiM05SSVVwTUZ1YmR
Xa1I3L3EwNXJJWG92OXcrSDRieDJZUGZyNzFEcVExNnB1d0hmWk01QkVWTUp6OEZYR0gvU2lWSW5SYVB
Rdi9hMXp3Tm1EeWdCRjlqU0VUZ1NNempvc2grN2FoM1VLSlZ5UEdzL3Q0NkNlSWg5LzNTd3lxQlJ0VjM
4VStsZzRzT3ZUK2pJMjByRXhlM3hudXI0cys4akt6TTlwWkdPalJmZHY3MHMraEMzTXJuZzNOT21ERXU
xQUZCcGQvdncwTnppano4WFpmS3lNQT09IiwidXNlcl9uYW1lIjoiNDQwNDQ3NzIwc3AiLCJzY29wZSI
6WyJyZWFkIl0sInVzdWFyaW8iOiJMUTlMbDZVcWFJSERYOWJOYWRqWWM5TlJkdnpDdHFGSEN6NHpYbVd
Qbm1IbCtCcFhYVEhCdlB5bkJMU3MxZHlXa2w1M2JIYUdON0xMUzZlV2hUeHFNVFhJcWg3SE1QVG1jNTN
mek5KOUd5bWo5cFNaMG1GdUN5RmEyM2JiZHdvWkphZzdiM3h1cGVBUUNWSmJ2OVZRYUd6UXljMjkzREd
yUFZDVjFEeVM2bXRmSnptelVyRXYrQjQrYkFjYkZpMUpnY3RERXBGb0NTT0tkMXNoMjhEaklCUytPS3l
xKzRMckl1MjdIRThEOUt0d01aWnF5bWFqZzhZMHZwWTNxcnJPMHRIbXBacytoVzlrL1BYUk9TNnR4R05
ZdEU0LzNsWTFCNnhtQ244RjlnTEhZTFNCNmlyUStHMllCUFNiNkV0Mnd2bzBo132V0g2MVIreXJzNERW
OFFQcEE9PSIsImV4cCI6MTQ5Mzg3OTYyMSwianRpIjoiOGMyNzQ4YzYtMzlhMi00MmY2LWI2OGQtNzA5
MzU1Njc2ZmRlIiwiY2xpZW50X2lkIjoiZGVtbyJ9.d6UedVr4TLiY0Dl_NrujVSlVbJsOBI7lpR7Lble
RontvBfQ7IjE2aWh0UETQwXfA4V2XbAw_3x9PjqtMZR6Skltdbpkqn Q6_GOVpnSaZu1XsSyR4tNLByHp
VKblVTq30T4W3VawBPZTH94_rW-4TLTuVdDF0qeqwKiDmDr4-
wIU4lbkLnDbyNsNyvKuUcTDiEzj83cOE9r2y_LQj_-
Ma44zVIayHLbAMCbE9XwhmzQPUqBTw534aFuXsGu7H3FwmUdYzeJ60LzzWtqwfG8OHoZTJMrTz3ak8eF
3o6Ow9LNWhZ4AvBMhEnmqSJOcHYVhndFi8KkJt_g34Iw35N-PEpA"
}
```


c. Salve o conteúdo do atributo **access_token**

```
$ access_token
="eyJhbGciOiJSUzI1Ni123IsInR5cCI6IkpXVCJ9.eyJzZW5oYSI6Ikh0V0JFVmF2ZG1UdnkvcGRYbT
FmaldZekQ0N2xqRWRFWUpsOG45VnlsVUZSMy9nYUk4VkF2NlFWdUhrR3BFQmlRSXZiNjFrMGdHQW4vN1
g3aXE3ODdPM0JVOGsyQ2llOGZNSlhJSlYyM2orbDU1V2dJNmhZODVSaUZTek40R0ZiM05SSVVwTUZ1Ym
RXa1I3L3EwNXJJWG92OXcrSDRieDJZUGZyNzFEcVExNnB1d0hmWk01QkVWTUp6OEZYR0gvU2lWSW5SYV
BRdi9hMXp3Tm1EeWdCRjlqU0VUZ1NNempvc2grN2FoM1VLSlZ5UEdzL3Q0NkNlSWg5LzNTd3lxQlJ0Vj
M4VStsZzRzT3ZUK2pJMjByRXhlM3hudXI0cys4akt6TTlwWkdPalJmZHY3MHMraEMzTXJuZzNOT21ERX
UxQUZCcGQvdncwTnppano4WFpmS3lNQT09IiwidXNlcl9uYW1lIjoiNDQwNDQ3NzIwc3AiLCJzY29wZS
I6WyJyZWFkIl0sInVzdWFyaW8iOiJMUTlMbDZVcWFJSERYOWJOYWRqWWM5TlJkdnpDdHFGSEN6NHpYbV
dQbm1IbCtCcFhYVEhCdlB5bkJMU3MxZHlXa2w1M2JIYUdON0xMUzZlV2hUeHFNVFhJcWg3SE1QVG1jNT
Nmek5KOUd5bWo5cFNaMG1GdUN5RmEyM2JiZHdvWkphZzdiM3h1cGVBUUNWSmJ2OVZRYUd6UXljMjkzRE
dyUFZDVjFEeVM2bXRmSnptelVyRXYrQjQrYkFjYkZpMUpnY3RERXBGb0NTT0tkMXNoMjhEaklCUytPS3
lxKzRMckl1MjdIRThEOUt0d01aWnF5bWFqZzhZMHZwWTNxcnJPMHRIbXBacytoVzlrL1BYUk9TNnR4R0
5ZdEU0LzNsWTFCNnhtQ244RjlnTEhZTFNCNmlyUStHMllCUFNiNkV0Mnd2bzBo132V0g2MVIreXJzNER
WOFFQcEE9PSIsImV4cCI6MTQ5Mzg3OTYyMSwianRpIjoiOGMyNzQ4YzYtMzlhMi00MmY2LWI2OGQtNzA
5MzU1Njc2ZmRlIiwiY2xpZW50X2lkIjoiZGVtbyJ9.d6UedVr4TLiY0Dl_NrujVSlVbJsOBI7lpR7Lbl
eRontvBfQ7IjE2aWh0UETQwXfA4V2XbAw_3x9PjqtMZR6Skltdbpkqn Q6_GOVpnSaZu1XsSyR4tNLByH
pVKblVTq30T4W3VawBPZTH94_rW-4TLTuVdDF0qeqwKiDmDr4-
wIU4lbkLnDbyNsNyvKuUcTDiEzj83cOE9r2y_LQj_-
Ma44zVIayHLbAMCbE9XwhmzQPUqBTw534aFuXsGu7H3FwmUdYzeJ60LzzWtqwfG8OHoZTJMrTz3ak8eF
3o6Ow9LNWhZ4AvBMhEnmqSJOcHYVhndFi8KkJt_g34Iw35N-PEpA"
```

- Realize a consulta na API
  a. Faça a consulta

```
$ curl -X GET -H "Authorization: Bearer ${access_token}"
http://api.sigapi.info/api/dados-cadastrais
```

  b. Verifique o resultado

```
{
  "nome": "Juca da Silva",
  "foto":
"https://sigacentropaulasouza.com.br/imagem/Z33YVAVR4NXTBKX8UJL6X9KMN35SR5.TMB.J
PG",
  "ra": "98657842147",
  "instituicao": "Faculdade de Tecnologia de São Bernardo do Campo \"Adib Moises
Dib\"",
  "curso": "Tecnologia em Informática para Negócios",
  "turno": "Noturno",
  "emailFatec": "nome.sobrenome@fatec.sp.gov.br",
  "emailWebsai": "nome.sobrenome@gmail.com",
  "emailPreferencial": "nome.sobrenome@gmail.com",
  "outrosEmails": [
    "nome.sobrenome@outlook.com",
    "nome.sobrenome@yahoo.com.br"
  ]
}
```

Postman

1. Crie uma nova requisição
   - **Método**: GET
   - **URL**: http://api.sigapi.info/api/dados-cadastrais
2. Configure a autenticação para obter o token
   a. Clique na aba **Authentication**
   b. Selecione o valor **OAuth 2.0** no campo **Type**
   c. Clique no botão **Get New Access Token**
   d. Preencha os dados solicitados
      - **Token Name**: sigapi token
      - **Auth URL**: http://login.sigapi.info/oauth/authorize?client_id=exemplo&redirect_uri=https://www.getpostman.com/oauth2/callback&response_type=code
      - **Access Token URL**: http://login.sigapi.info/oauth/token?redirect_uri=https://www.getpostman.com/oauth2/callback
      - **Client ID**: exemplo
      - **Client Secret**: exemplo
      - **Grant Type**: Authorization Code
   e. Clique no botão **Request Token**
      i. Na tela que se abre, preencha seu usuário e senha e clique no botão **login**
         - Aguarde uns instantes, pois o Postman não exibe nenhuma informação de que a tela está sendo carregada
      ii. Na tela de autorização, clique no botão **Authorize**
3. Configure a requisição para utilizar o token obtido
   a. Clique na linha referente ao token chamado **sigapi token**
   b. No lado direito, realize a configuração
      - No campo **Add token to** selecione o valor **Header**
      - Clique no botão **User Token**
4. Realize a requisição
   a. Clique no botão **Send**, localizado no canto superior direito
5. Verifique o resultado
   a. Na parte inferior da janela será exibido o JSON de retorno

Spring Boot

O exemplo a seguir demonstra apenas a configuração do uso da API em um projeto já existente e funcionando

- Adicione as dependências para uso do Spring Security OAuth
  a. Maven

```
<dependency>
    <groupId>org.springframework.security.oauth</groupId>
    <artifactId>spring-security-oauth2</artifactId>
</dependency>
```

- Habilite o cliente oauth2 adicionando a anotação `@EnableOAuth2Client` em qualquer classe de configuração
- Crie um `Bean` configurando o `RestTemplate` a ser utilizado

```
@Bean
public RestTemplate restTemplate(final OAuth2ClientContext context) {

    final AuthorizationCodeResourceDetails resource = new
AuthorizationCodeResourceDetails();
    resource.setClientId("exemplo");
    resource.setClientSecret("exemplo");
    resource.setAccessTokenUri("http://login.sigapi.info/oauth/token");
    resource.setUserAuthorizationUri("http://login.sigapi.info/oauth/authorize");

    return new OAuth2RestTemplate(resource, context);

}
```

- Injete o `RestTemplate` na classe onde irá utilizá-lo

```
@Autowired
private RestTemplate restTemplate;
```

- Realize a chamada ao serviço desejado

```
public void metodo() {

    final JsonNode dadosCadastrais = restTemplate.getForObject
("http://api.sigapi.info/api/dados-cadastrais", JsonNode.class);

}
```

PHP

O exemplo a seguir demonstra um fluxo completo de autorização, obtenção do token e obtenção dos dados cadastrais. Só copiar, colar e usar

```php
<?php

session_start();

$accessToken = null;
if(!empty($_SESSION['accessToken'])) {
    $accessToken = $_SESSION["accessToken"];
}

// Verifica se possui o token de acesso
if ($accessToken) {

    // Já possui o token, irá requisitar os dados cadastrais
    $process = curl_init();
    curl_setopt($process, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($process, CURLOPT_TIMEOUT, 120);
    curl_setopt($process, CURLOPT_URL, "http://api.sigapi.info/api/dados-cadastrais");
    curl_setopt($process, CURLOPT_HTTPHEADER, array("Authorization: Bearer
$accessToken"));
    $result = curl_exec($process);
    $resultCode = curl_getinfo($process, CURLINFO_HTTP_CODE);
    curl_close($process);
    $jsonResult = json_decode($result);

    if ($resultCode == "200") {
        echo "Nome: $jsonResult->nome <br>";
        echo "RA: $jsonResult->ra <br>";
        echo "Instituição: $jsonResult->instituicao <br>";
        echo "Curso: $jsonResult->curso <br>";
    } else {
        echo "Ocorreu um erro ao acessar a API: $resultCode / $jsonResult->mensagem";
    }

} else {

    // URL atual
    $currentUrl = strtok("http://$_SERVER[HTTP_HOST]$_SERVER[REQUEST_URI]", "?");

    // Verifica se possui o código de autorização
    $authorizationCode = $_GET["code"];
    if ($authorizationCode) {

        // Irá obter o token de acesso
        $accessTokenUrl =
```

```
"http://login.sigapi.info/oauth/token?code=$authorizationCode&grant_type=authorization
_code&redirect_uri=$currentUrl";

        $process = curl_init();
        curl_setopt($process, CURLOPT_POST, 1);
        curl_setopt($process, CURLOPT_RETURNTRANSFER, true);
        curl_setopt($process, CURLOPT_TIMEOUT, 120);
        curl_setopt($process, CURLOPT_URL, $accessTokenUrl);
        curl_setopt($process, CURLOPT_USERPWD, "exemplo:exemplo");
        $result = curl_exec($process);
        curl_close($process);
        $jsonResult = json_decode($result);

        if ($jsonResult) {

            if(!empty($jsonResult->error)) {
                echo "Houve um erro obtendo o Token: $jsonResult->error ($jsonResult-
>error_description)";
            } else {

                // Salva o token na sessão e recarrega a página, para reiniciar o
processamento
                $_SESSION['accessToken'] = $jsonResult->access_token;
                header("Location: $currentUrl");

            }

        } else {
            echo "Ocorreu algum erro ao tentar obter o token de acesso";
        }

    } else {

        // Como não possui o código de autorização, redireciona para que o usuário
possa se conectar
        $authorizationUrl =
"http://login.sigapi.info/oauth/authorize?client_id=exemplo&redirect_uri=$currentUrl&r
esponse_type=code";
        header("Location: $authorizationUrl");

    }

}

?>
```