InternSheep: Sistema *Web* para Divulgação de Vagas de Estágio/Trainee à Comunidade Discente do IFSP-HTO

Rodrigo C. Salvadori, Michele Cristiani Barion

Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas

Instituto Federal de São Paulo – Câmpus Hortolândia

r.salvadori@aluno.ifsp.edu.br, michelefreitas@ifsp.edu.br

Abstract. *This paper presents a system development for the dissemination of internship/trainee vacancies within the IFSP, and was motivated by the method of publicizing the internship/trainee vacancies that are offered within the IFSP. Currently, e-mails and messages are used in the process of publicizing vacancies on the SUAP, in addition to internal dissemination on bulletin boards on campus. It is shown that the insertion of the student in the job market is important both during the course period and at its conclusion, so that the student can exercise the obtained knowledge. At the end of the paper, a back-end services API was developed, where it is able to receive JSON requests from other microservices, and this entire system was developed according to the agile methodology.*

Resumo. *O presente artigo apresenta o desenvolvimento de um sistema para a divulgação de vagas de estágio/trainee dentro do IFSP, e teve como motivação o método de divulgação das vagas de estágio/trainee que são ofertadas dentro do IFSP. Atualmente são utilizadas no processo de divulgação das vagas e-mails e mensagens no SUAP, além da divulgação interna em quadros de aviso nas dependências do Câmpus. Mostra-se que a inserção do aluno no mercado de trabalho é importante tanto no período do curso quanto em sua finalização, para que o mesmo possa exercer o conhecimento obtido. Ao final do artigo, foi desenvolvida uma API de serviços back-end, onde a mesma está apta a receber requisições JSON de outros microserviços, tendo todo esse sistema desenvolvido de acordo com a metodologia ágil.*

1. Introdução

Este trabalho teve como motivação o método de divulgação das vagas de estágio/trainee que são ofertadas dentro do IFSP. Atualmente são utilizadas no processo de divulgação das vagas e-mails e mensagens no SUAP, além da divulgação interna em quadros de aviso nas dependências do Câmpus.

A inserção do estudante no mercado de trabalho é de grande importância para que o mesmo possa exercitar o conhecimento obtido durante o período de graduação. Dito isso, é necessário que o estudante tenha fácil acesso às oportunidades oferecidas no prazo estabelecido na vaga com todas as informações que são necessárias para sua inscrição nas oportunidades que são ofertadas.

Com a conclusão desta proposta, espera-se um canal de comunicação mais ativo e participativo com os estudantes que almejam uma oportunidade na área de formação e, assim, haja uma aproximação com o mercado de trabalho.

O presente trabalho está organizado em seis Seções ordenadas em: fundamentação teórica, metodologia, desenvolvimento, conclusões, trabalhos futuros e relação das referências utilizadas tanto na pesquisa sobre as metodologias adotadas quanto das ferramentas que foram fundamentais para o desenvolvimento.

2. Referencial Teórico

Nesta Seção são abordados os principais conceitos que fundamentam os aspectos técnicos e de negócio quanto ao desenvolvimento deste trabalho.

2.1. Metodologias Ágeis

[SOMMERVILLE 2011] destaca em sua obra que metodologias ágeis são processos de desenvolvimento rápido e capazes de lidar com mudanças nos requisitos de *software*. Este processo visa a produção mais rápida de *softwares* utilizáveis, antes mesmo que todas as funcionalidades do sistema estejam prontas. Isso faz com que a aplicação seja desenvolvida em uma série de incrementos com novas funcionalidades a serem adicionadas no sistema, ao invés do desenvolvimento em uma única unidade.

2.2. *Test-Driven Development* ou Desenvolvimento Guiado por Testes (*TDD*)

TDD (Test-Driven Development) ou Desenvolvimento Guiado por Testes é uma metodologia de desenvolvimento de *software* descoberta por Kent Beck que também criou a metodologia ágil XP (Xtreme Programming) onde essa se relaciona conceitualmente com o TDD.

O *TDD* parte do princípio que, primeiramente, é necessária a escrita de um teste unitário de acordo com os requisitos da aplicação, para que seja feita escrita do próprio código satisfazendo a condição do teste [ANICHE 2012].

2.3. Levantamento de requisitos

Como apresenta o autor [Sommerville 2011], os requisitos de um sistema são as descrições dos serviços e as restrições de sua operação. Esses requisitos refletem as necessidades dos clientes de um sistema que atende um determinado propósito, como controlar um dispositivo, fazer um pedido ou encontrar informações. Há diversas ferramentas para coletar os requisitos através dos *stakeholders*, sendo que os mais usados são o questionário e a entrevista.

Se os requisitos funcionais são aqueles que o sistema deve ser capaz de executar, os requisitos não funcionais são as restrições em serviços ou em funções oferecidas pelo sistema, como por exemplo, restrições de tempo, no processo de desenvolvimento ou impostas por normas [SOMMERVILLE 2011].

2.4. Angular JS

AngularJS é um *framework* estrutural JavaScript mantido pelo Google, tendo como propósito auxiliar na construção de páginas interativas, permitindo a inclusão de regras e lógicas no *frontend*, contornando o cenário de várias requisições ao servidor *backend*.

Através de um *framework* JavaScript torna os métodos executados no *frontend* mais simples e claros, abstraindo complexidades desnecessárias, facilitando e agilizando o desenvolvimento de aplicações [ANGULARJS 2021].

2.5. Spring Framework

Como apresenta [DEVMEDIA 2021], o Spring Framework surgiu em meados do ano 2002, sendo *open source* e sua criação teve a finalidade de possibilitar a construção de aplicações desenvolvidas em Java.

2.6. JavaScript

JavaScript é uma linguagem lançada em meados da década de 90, tendo como propósito trazer interatividade às páginas estáticas da *web*. "É uma linguagem desenvolvida para rodar no lado do cliente, isto é, a interpretação e o funcionamento da linguagem dependem de funcionalidades hospedadas no navegador do usuário. Isso é possível porque existe um interpretador *JavaScript* hospedado no navegador" [SILVA 2010].

A linguagem possibilita a interação com *HTML* e com o conteúdo da página sem a necessidade de recarregar a mesma. Também é possível tratar de cliques ou outros eventos e, assim, ampliar e melhorar a relação com o usuário.

Como apresenta [SILVA 2010], outro aspecto positivo desta linguagem é que pode ser usada em conjunto com outras linguagens para executar tarefas complementares que estão relacionadas ao fluxo da programação.

2.7. *JavaScript Object Notation (JSON)*

JavaScript Object Notation (JSON) é um formato baseado em texto padrão para representar dados estruturados com base na sintaxe do objeto JavaScript. Não possui métodos, apenas propriedades do objeto. Um exemplo da sua aplicação é a transmissão de dados por meio de aplicativos da *web*.

Assemelha-se à sintaxe literal do objeto *JavaScript*, podendo incluir os mesmos tipos de dados, tais como strings, números, booleanos, matrizes e outros literais de objeto [SMITH 2015].

A linguagem Java traz bibliotecas para trabalhar com a integração de dados no formato *JSON*.

2.8. Banco de Dados Relacional

Segundo [ELMASRI e NAVATHE 2018], um banco de dados é uma coleção de dados logicamente relacionados, sendo esses associados às características do mundo real denominados Mini-Mundo ou Universo de Discurso. Um banco de dados pode ser criado e mantido pelos Sistemas de Gerenciamento de Banco de Dados (SGBD). O SGBD é um *software* que facilita o processo de definição e especificação dos tipos de dados, suas estruturas e restrições, denominados metadados; construção; armazenamento e manipulação; funções de consulta, e por fim o compartilhamento de banco de dados que permite o acesso simultâneo entre usuários e aplicações [HEUSER 2009].

No modelo relacional as tabelas são compostas por linhas que representam uma coleção de valores de dados. Para armazenar os dados neste modelo é necessário especificar as propriedades de cada campo que identifica um valor, além do cumprimento de restrições de integridade. É importante enfatizar que esse modelo é chamado de relacional devido à relação que há entre os campos denominados chave primária (*primary key*) e chave estrangeira (*foreign key*) e, assim, cumprindo as regras de normalização para manter a integridade e evitar redundância de dados [ELMASRI E NAVATHE 2018].

Os bancos de dados relacionais oferecem linguagens e interfaces para criação e manipulação de uma base de dados. A linguagem padrão utilizada nos bancos de dados desse modelo é a *Structured Query Language*, mais conhecida como *SQL*.

Como apresenta [ELMASRI e NAVATHE 2018] e conforme abordagem para este trabalho, a linguagem *SQL* associa conjuntos de comandos em algumas camadas, a saber:

- *DDL* é chamada de Linguagem de Definição de Dados e associa comandos *SQL* para criação dos esquemas de banco de dados: CREATE, DROP, TABLE.
- *DML* é chamada de Linguagem de Manipulação de Dados e executa comandos para inserção, seleção, modificação ou remoção de dados: INSERT, SELECT, UPDATE e DELETE.

3. Trabalhos Correlatos

Cita-se o trabalho do autor [Salvino 2019] que teve como objetivo o desenvolvimento de uma aplicação para gerenciamento de estágios do IFSP do Câmpus Hortolândia, a qual alguns conceitos abordados por ele fundamentaram algumas partes da implementação da proposta deste trabalho. [Salvino 2019] não focou o seu trabalho na divulgação das vagas, mas sim no gerenciamento de alunos e seus estágios, considerando documentos a serem entregues, tais como relatórios mensais de estágio, contratos, rescisão, entre outros associados a essa relação de escola e empresa. Assim, surge a ideia que fundamenta a proposta de divulgar as vagas de estágio e com a possibilidade futura de fazer uma integração entre essa aplicação e a proposta deste trabalho.

4. Metodologia

A proposta é um estudo de caso tendo como cenário o IFSP Câmpus Hortolândia. O levantamento de requisitos se dá por meio de reuniões com a Coordenadoria de Extensão, sendo esta responsável pela divulgação das vagas, como também os orientadores de estágio que poderão agregar na definição das funcionalidades da aplicação.

O desenvolvimento do trabalho é apresentado em *Sprints*, de acordo com o que é proposto na Metodologia Ágil, utilizando para a escrita dos testes e casos de uso o modelo de desenvolvimento *TDD*.

Quanto aos materiais para desenvolvimento:

- Para desenvolvimento da aplicação é utilizado *IDE IntelliJIDEA Community Editon* fornecida pela *JetBrains*, considerando a linguagem de programação Java para o *backend*.

- Para o *frontend* é utilizado o *framework Angular JS* em conjunto com o *Angular Material*, sendo fundamentais para o *design* da proposta, além do Figma que é uma ferramenta para *design* de interface.

- Para integração do *frontend* e *backend* são feitas requisições através do *JSON* para o fornecimento dos dados obtidos na base de dados, sendo essa modelada de acordo com a regra de negócio. O armazenamento de dados segue o modelo relacional com o uso da linguagem *SQL*.

5. Desenvolvimento

Esta Seção destaca o atual desenvolvimento do trabalho, dividindo-o em subseções contendo as histórias de usuários definidas juntamente com as *Sprints* em que serão realizadas e os testes que foram propostos dentro da metodologia *TDD*, mostrando esses passos para que se possa melhor entender o processo de análise e desenvolvimento do projeto.

5.1. Levantamento de requisitos e diagramas

Assim como foi definido no escopo deste trabalho, foram levantados os requisitos não funcionais e funcionais do sistema, mostrados na Tabela 1 e Tabela 2 respectivamente. Conforme exposto na metodologia, enfatiza-se que os requisitos foram levantados através de reuniões no ambiente de estudo, tendo sido registrados cinco encontros. Os *stakeholders* apresentaram o que o Câmpus teria de estrutura física para implantar a aplicação, como também como é feita a divulgação de vagas à comunidade acadêmica. Essas informações foram fundamentais para buscar a informatização dos processos apresentados.

Tabela 1. Requisitos não funcionais

RNF	Descrição
RNF-1	Servidor do banco de dados e web para executar a aplicação que deve ser implantado no Câmpus. O acesso será feito através de *link* externo, estando implantado na estrutura do Câmpus para poder ter seu acesso a qualquer hora. (requisito de disponibilidade).
RNF-2	Segurança oferecida pelo domínio do *site* do campus. Devido a estar hospedado no domínio exclusivo do IFSP-HTO, poderá ser acessado somente por pessoas do Câmpus (requisito de segurança).

Tabela 2. Requisitos funcionais

RF	Descrição
RF-1	Acessar sistema de vagas através de login com acesso ao servidor que estará hospedado no Câmpus. Requisito para todos os tipos de usuários.
RF-2	Visualizar vagas com login efetuado. Requisito para todos os tipos de usuários.
RF-3	Ver descrição das vagas através da lista geral, direcionando para uma página contendo os detalhes da mesma. Requisito para todos os tipos de usuários.
RF-4	Cadastrar novas vagas. Requisito restrito para o usuário servidor.
RF-5	Verificar validade da vaga para apresentar na listagem geral.

A Figura 1 apresenta o diagrama do banco de dados, considerando as tabelas: usuarios, historico_vagas e vaga.

A tabela usuarios contém os campos de id_usuario, nome_usuario, user_name e senha que irão armazenar as futuras informações de *login*, assim que essa funcionalidade for implementada. Já a tabela vaga, contém os campos: id_vaga para garantir a diferenciação das vagas, nome, descricao, conteudo, tipo_vaga, data_de_vencimento e link_vaga. Essa tabela é a principal do sistema, todas as informações sobre as vagas anunciadas serão armazenadas. E a tabela historico_vaga contém os mesmos campos que a tabela vaga, pois ela irá funcionar como um histórico de todas as vagas que foram anunciadas dentro do sistema, sendo alimentada pela função de validação de vagas, detalhada na Seção 5.4 deste artigo.

Para a construção desse diagrama como também o gerenciamento da base de dados foi utilizado a ferramenta de gerenciamento PHPMyAdmin.

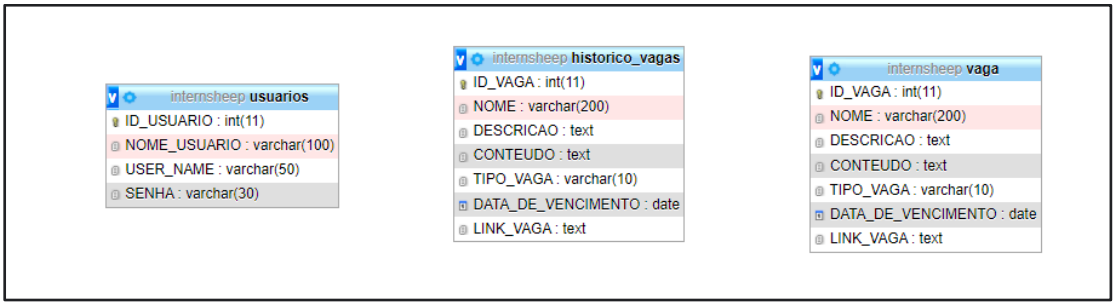

Figura 1. Diagrama do Banco de Dados

A Figura 2 traz as funcionalidades propostas para o sistema por meio do diagrama de caso de uso, considerando os requisitos funcionais apresentados na Tabela 2. A partir dessa

Figura é mostrado o fluxo de atividades que podem ser executadas no sistema, com os atores sendo o próprio usuário e estando distintos entre Discente e Servidor, sendo a funcionalidade de Anunciar vagas limitada ao ator do tipo Servidor. Seguindo o fluxo de atividades, o Usuário irá Efetuar login no sistema e com isso, é redirecionado para a função do Listar vagas, onde são obtidas todas as vagas que estão cadastradas no banco de dados, e ficando por decisão dele ou não de usar a função de Detalhar vaga para obter os detalhes específicos de uma única vaga.

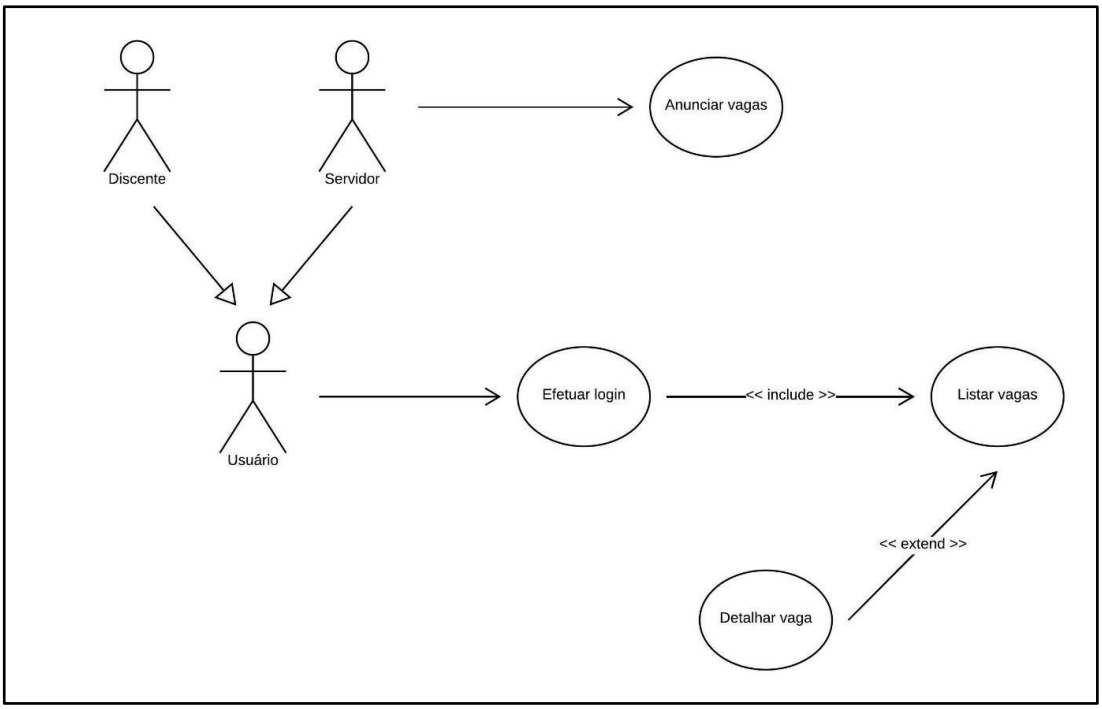

Figura 2. Diagrama de Caso de Uso

A Figura 3 mostra a organização da arquitetura do sistema, abordando a teoria de microsserviços. Enfatiza-se que as funções do sistema são separadas em serviços diferentes para melhorar a performance e a manutenção do sistema. No caso desse sistema, a arquitetura foi dividida em 3 serviços, sendo eles: *Frontend API*, *Login API* e *API* Vagas. A *Frontend API* é onde toda a estrutura e código referente ao *frontend* (*design* e código visível ao usuário no qual a página do sistema poderá ser renderizada) está contida, assim deixando separado do *backend* para que haja uma melhor leitura de código. Tal como o nome diz, o microserviço *Login API* é responsável por toda parte de acesso ao sistema, independente dos outros componentes. Com isso, garantimos que tudo relacionado ao *login* fique isolado do restante do sistema, controlando o acesso às suas funcionalidades, com o acesso ao banco de dados estabelecido para procurar as informações dos usuários. E, por fim, o último microserviço que contém toda a regra de negócio para as funcionalidades definidas no diagrama de caso de uso, conforme apresentado na Figura 2. Toda a comunicação entre os microserviços é feita através de requisições *HTTP,* a qual cada um deles terá seu próprio endereço para recebimento dessas requisições, e para o retorno das informações entre a comunicação dos serviços.

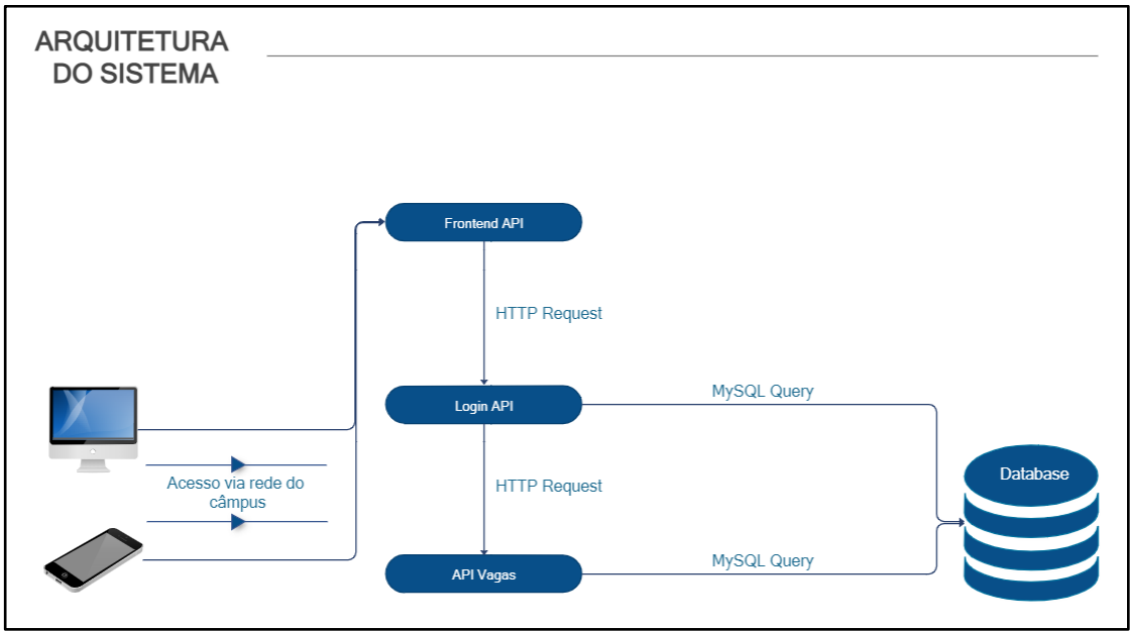

Figura 3. Diagrama de Arquitetura do Sistema

Já na Figura 4, é mostrado o Diagrama de Classes do sistema, que foi gerado pela ferramenta de diagramação de sistema na *IDE* IntelliJ IDEA. O diagrama mostra como está estruturado o *backend* da aplicação, detalhando as classes que foram construídas, seus atributos, métodos e propriedades. Para esse sistema foram definidas classes de modelo que são as classes responsáveis por armazenar os dados retornados na consulta ao banco de dados, tendo a nomenclatura similar à sua modelagem (classe Vaga e Usuario), como apresentada na Figura 1. As classes com nomenclatura de DAO e JDBC são responsáveis pela definição e acesso ao banco de dados. Dessas classes, as que estão sendo declaradas como do tipo DAO, contém as assinaturas dos métodos de acordo com a regra de negócio para se obter as informações. Já as classes que estão sendo declaradas como do tipo JDBC, contém a implementação desses métodos e o devido acesso e execução das *queries* no banco de dados, com o retorno destas das informações obtidas nos métodos dessas classes para as classes do tipo/nomenclatura *Service*, que irão manipular os dados e aplicar as regras de negócio para retornar essas informações para a classe ApplicationController. Essa classe é a que deve transformar o código em uma *API*, declarando todos os *endpoints* necessários para as requisições *HTTP* e o tratamento dessas requisições para suas possíveis exceções *HTTP*, como quando retornar código de erro 400, 404, entre outros. Além das classes apresentadas, outras são responsáveis pela configuração da *API*, como a WebConfig por indicar a criação do *bean* de acesso ao banco de dados, e a SpringBootJspApplication, responsável por dar o *start* na aplicação.

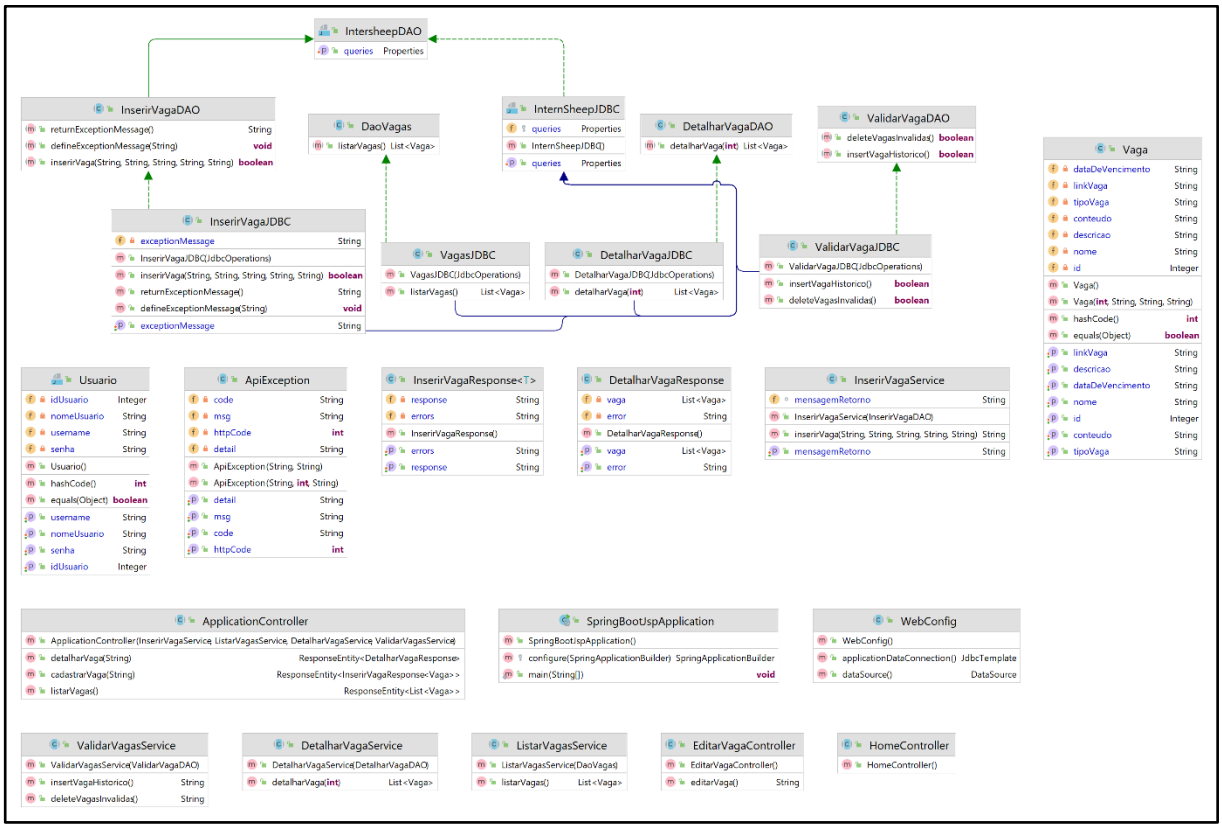

Figura 4. Diagrama de Classes

5.2. *Sprints*

A Tabela 3 mostra a definição das *Sprints* e suas correspondentes atividades, respeitando o tempo máximo de 4 semanas para cada *Sprint*, estando esse intervalo de tempo alinhado com o cronograma definido na proposta desse trabalho.

Tabela 3. Definição das *Sprints*

Número da *Sprint*	**Atividades**
Sprint 1	Escrita dos casos de testes *TDD*, para as funcionalidades relativas aos casos de uso mostrados na Figura 2: Listar vagas, Detalhar vaga e Anunciar vaga. Desenvolvimento e modelagem do banco de dados, e definição da arquitetura e tecnologias necessárias para o desenvolvimento.
Sprint 2	Desenvolvimento do código *backend* e aplicação dos casos de testes *TDD* em cenário positivo e negativo para a funcionalidade relativa aos casos de uso mostrados na Figura 2: Listar vagas.
Sprint 3	Desenvolvimento do código *backend* e aplicação dos casos de testes *TDD* em cenário positivo e negativo para a funcionalidade relativa aos casos de uso mostrados na Figura 2: Detalhar vaga.
Sprint 4	Desenvolvimento do código *backend* e aplicação dos casos de testes *TDD* em cenário positivo e negativo para a funcionalidade relativa aos casos de uso mostrados na Figura 2: Anunciar vaga.
Sprint 5	Desenvolvimento do código *backend* da rotina de verificar validade das vagas.
Sprint 6	Integração do código *backend*, desenvolvimento dos *templates frontend*.

5.3. Histórias de Usuário

Assim como está descrito na metodologia deste trabalho, após a análise e levantamento de requisitos e a construção dos casos de uso, foram definidas as histórias de usuário que irão compor as *Sprints*, assim como é mostrado na Tabela 4 e que devem ser trabalhadas dentro do prazo definido pelas mesmas.

Tabela 4. Histórias de Usuário

HU	Histórias de Usuário
HU-1	Como discente, quero acessar o sistema via login, inserindo usuário e senha como credenciais. (Tabela 2 – RF1)
HU-2	Como discente, quero visualizar as vagas de estágio/trainee disponíveis para verificar se tenho interesse em alguma vaga. (Tabela 2 – RF2)
HU-3	Como discente, quero visualizar o detalhe da vaga, para acessar todas as suas informações em seu anúncio completo. (Tabela 2 – RF3)
HU-4	Como servidor, quero conseguir anunciar vagas através da plataforma para divulgá-las no Câmpus. (Tabela 2 – RF4)

5.4. Templates *Frontend*

Serão apresentados nesta Subseção os *templates* das telas de cada funcionalidade. Sendo a proposta desenvolvida para plataforma *web*, as telas foram implementadas com o uso da ferramenta *opensource* Figma que é usada para criar *templates*, tendo sido desenvolvidos durante a *Sprint* 6.

A Figura 5 traz a tela de login da aplicação, considerando a história de usuário 1 (Tabela 4 - HU-1).

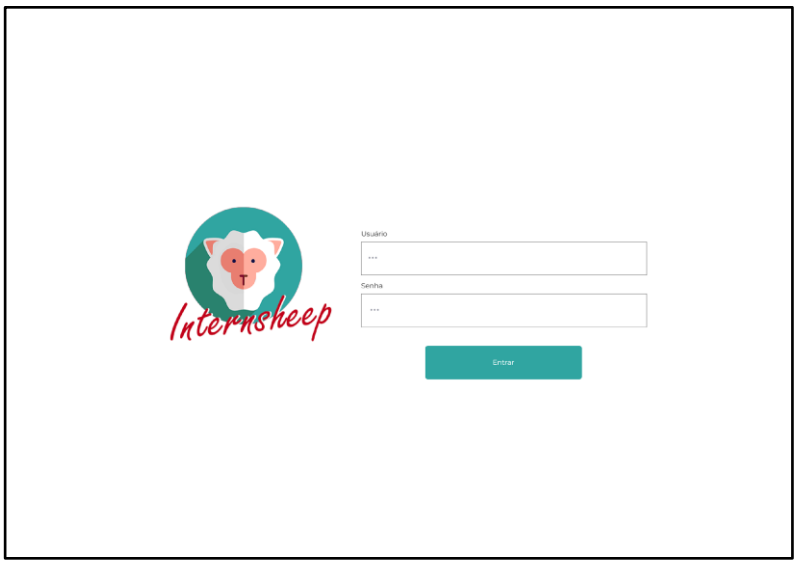

Figura 5. Tela de login da aplicação

A Figura 6 apresenta a tela para visualizar as vagas cadastradas após o login efetuado pelo usuário. Essa tela contempla a história de usuário 2 (Tabela 4 - HU-2).

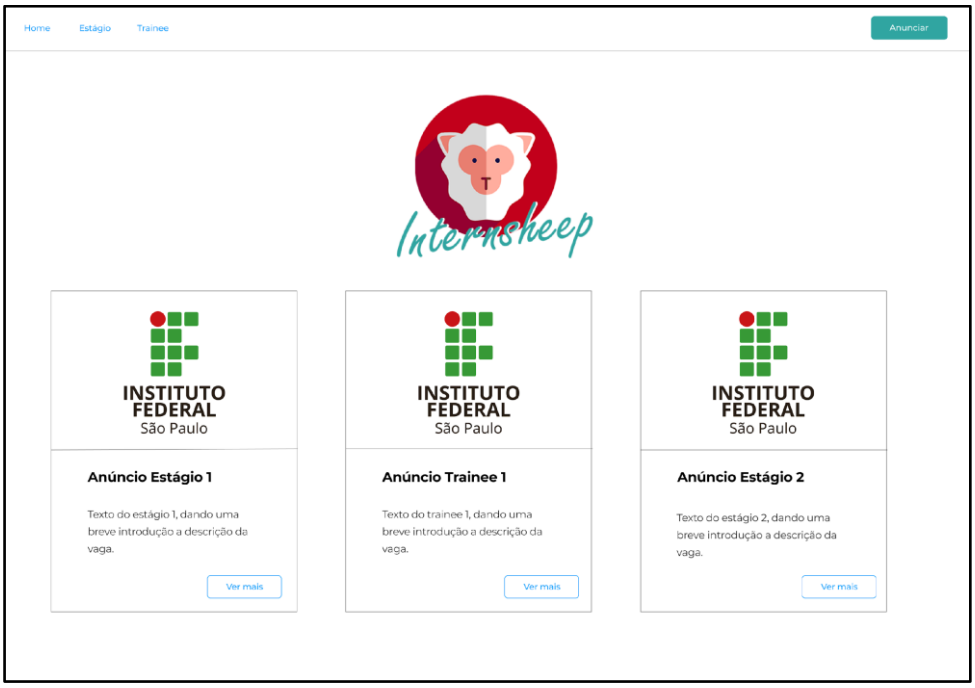

Figura 6. Tela de visualização de vagas

Já a Figura 7 tem como proposta trazer os detalhes da vaga, considerando a seleção pela lista cadastrada, como foi apresentada na Figura 4. Essa tela contempla a história de usuário 3 (Tabela 2 - HU-3).

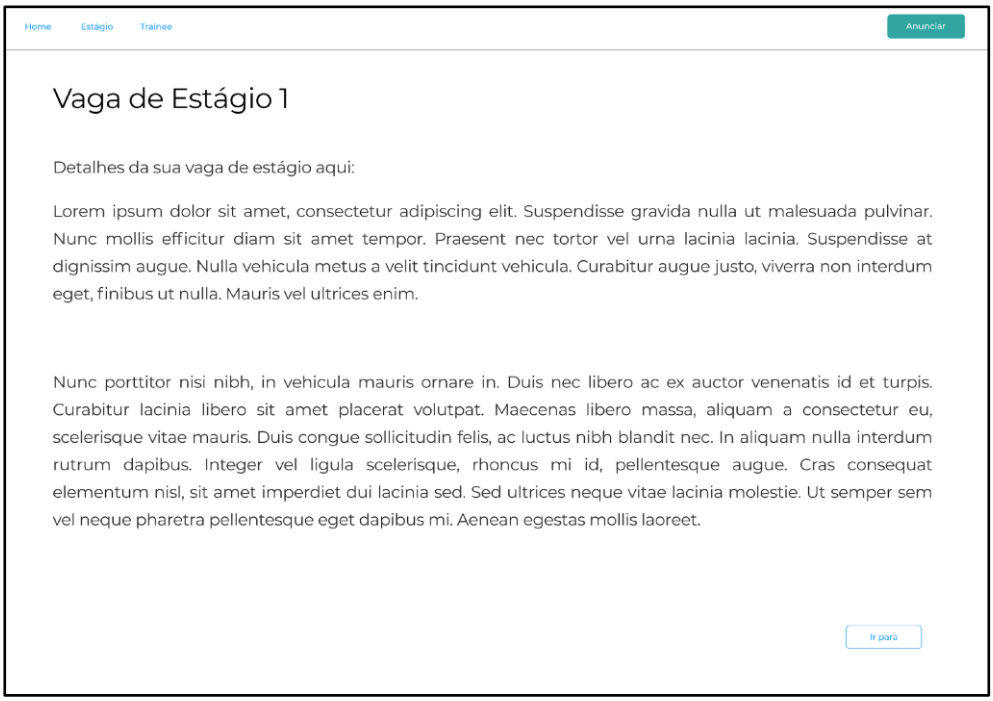

Figura 7. Tela de detalhamento da vaga selecionada

Considerando a história de usuário 4 (Tabela 4 - HU-4), a Figura 8 traz a funcionalidade de cadastro de vagas, sendo uma opção específica para o usuário denominado Servidor.

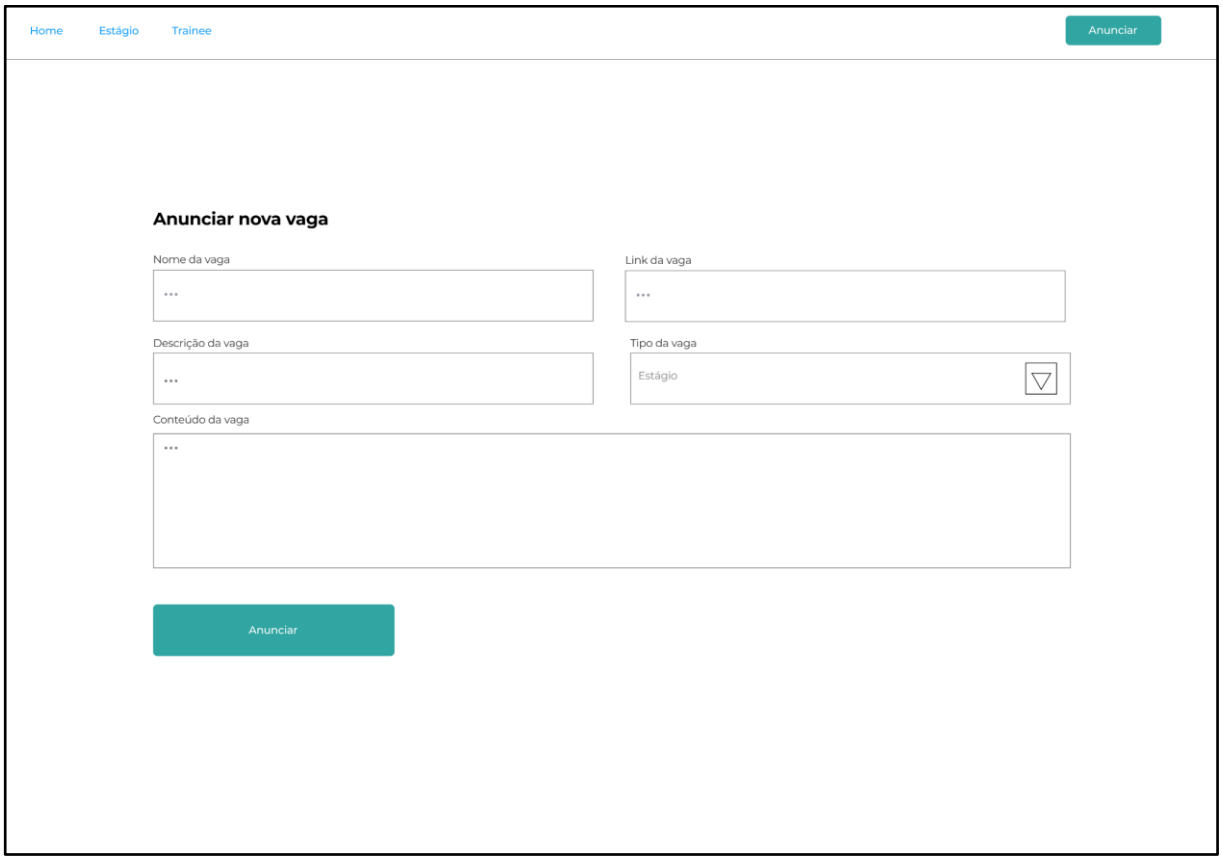

Figura 8. Tela para gerenciar novas vagas

Considerando o requisito funcional 5 (Tabela 2 - RF5), a Figura 9 traz a rotina implementada para analisar a validade das vagas cadastradas.

```
@RequestMapping(value = "/listarVagas", method = RequestMethod.GET)
public ResponseEntity<List<Vaga>> listarVagas() throws JsonParseException {
    List<Vaga> vagas = vagasService.listarVagas();
    validarVagasService.insertVagaHistorico();
    validarVagasService.deleteVagasInvalidas();
    return ResponseEntity.status(HttpStatus.OK).body(vagas);
}
```

Figura 9. Implementação da rotina para validação das vagas

Na rotina que foi desenvolvida durante a *Sprint* 5, mostrada na Tabela 3, são utilizados dois métodos que são executados sempre que é acessada a página inicial do sistema (*endpoint* /listarVagas), conforme o diagrama de caso de uso apresentado na Figura 2. Esses métodos são: *insertVagaHistorico* e *deleteVagasInvalidas,* sendo esses métodos responsáveis pelo gerenciamento de dados da tabela historico_vagas quanto às vagas que estão fora da data de validade. Para esse fim, considera-se a *query*:

```
INSERT INTO HISTORICO_VAGAS (ID_VAGA, NOME, DESCRICAO, CONTEUDO, TIPO_VAGA,
DATA_DE_VENCIMENTO, LINK_VAGA) SELECT * FROM VAGA WHERE DATA_DE_VENCIMENTO <
CURDATE().
```

Após a execução da consulta de inserção, as vagas denominadas vencidas, serão excluídas através da *query*:

```
DELETE FROM VAGA WHERE DATA_DE_VENCIMENTO < CURDATE().
```

A data de validade das vagas é definida em suas inserções no banco de dados, dando um prazo de 3 meses para ficarem disponíveis no sistema.

5.5. Testes

Esta Seção descreve quais testes foram definidos utilizando a metodologia *TDD* e trabalhando com cenários positivos e negativos para cada teste, trazendo suas descrições e seus resultados que foram satisfeitos ou não durante o desenvolvimento do sistema.

5.5.1. Visualização de Vagas

O teste para a visualização de vagas, consiste na principal função do sistema, tendo sido escrito na Sprint 1 e desenvolvido e testado na *Sprint* 2 como mostra a Tabela 3. De acordo com a arquitetura proposta, o teste irá ser satisfeito quando ao chamar a *URL* do sistema (que para todos os testes locais ficará hospedada na *URL*: http://localhost:9090) ele exiba todas as vagas disponíveis que foram devidamente cadastradas e estão armazenadas no banco de dados, como é exemplificado na Figura 4.

5.5.1.1. *Backend*

5.5.1.1.1. Caso de Teste: Cenário Positivo

A Tabela 5 traz a descrição do teste *backend* através da funcionalidade "listarVagas". Para o *backend*, ao acionar o *endpoint* "/listarVagas" que é o mesmo do acesso a *URL* http://localhost:9090/listarvagas, será executada uma requisição *HTTP*, que deverá consultar no banco de dados as vagas que estão armazenadas por meio da *query*: SELECT * FROM VAGA. A saída do resultado se dará no formato *JSON*. Esse *endpoint* é acionado através do *frontend* com o resultado tratado para apresentação como mostra a Figura 4.

Tabela 5. Teste do *endpoint* listarVagas (*backend*)

Teste	Ferramenta Utilizada	Resultado
Obter as vagas cadastradas no sistema	Insomnia	Sucesso

O teste foi realizado com a ferramenta Insomnia com requisições *HTTP*. A Figura 10 traz o teste da *API*.

Figura 10. Teste do *endpoint* listarVagas via Insomnia

Assim, ao realizar a requisição, é retornado um *JSON* contendo todas as vagas cadastradas no sistema e suas informações, como é mostrado na Figura 11.

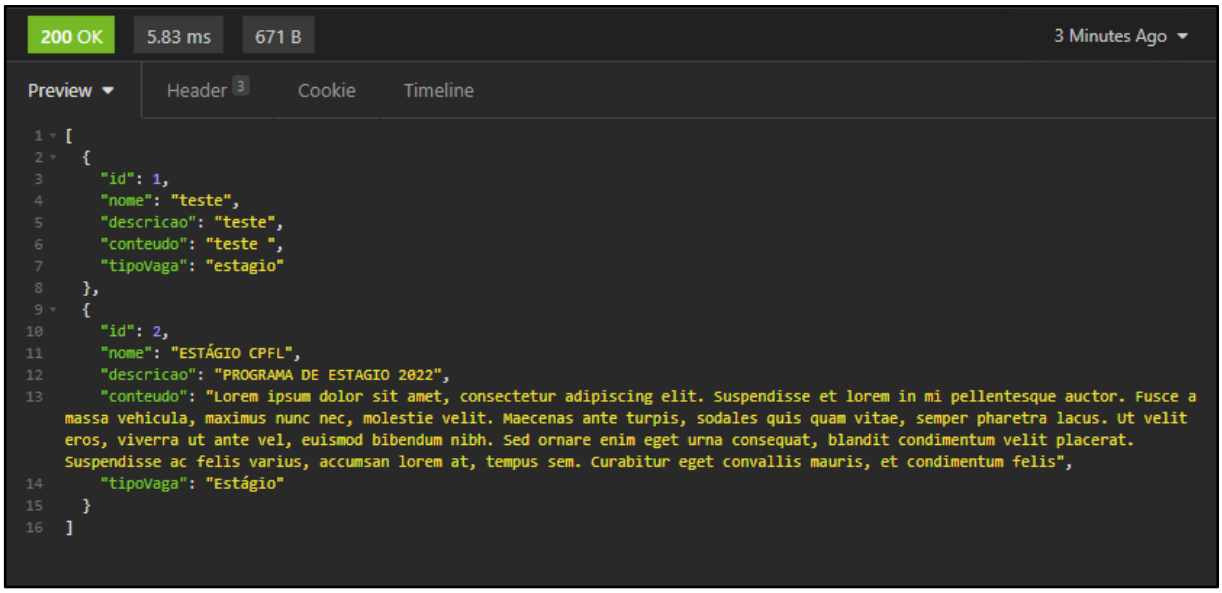

Figura 11. Retorno da requisição via Insomnia do *endpoint* listarVagas

No retorno da requisição é mostrado o *status* "200 OK" que significa que a requisição foi executada com sucesso. Assim, a resposta do *endpoint* para essa requisição serão os registros das vagas no banco de dados com seus campos que contém informações, inclusive seus *id's* únicos para as diferenciá-las.

5.5.1.1.2. Caso de Teste: Cenário Negativo

A Tabela 6 traz a descrição do teste *backend* com cenário negativo através da funcionalidade "listarVagas". Para este cenário, irá ser acionada a url *errada* "/listarVaga" que é o mesmo do acesso a *URL* http://localhost:9090/listarVaga, sendo executada uma requisição *HTTP*, que deverá consultar no banco de dados as vagas que estão armazenadas por meio da *query*: SELECT * FROM VAGA, mas neste caso, irá ocasionar erro devido ao caminho inválido acionado pela *URL*.

Tabela 6. Teste do *endpoint* listarVagas (*backend*)

Teste	Ferramenta Utilizada	Resultado
Forçar erro ao digitar *URL* errada	Insomnia	Sucesso

Portanto, ao digitar a *URL* de maneira incorreta, é esperado o erro 404 *Not Found,* que ocorre ao acessar um *endpoint* não mapeado pela *API*, como mostra a Figura 12.

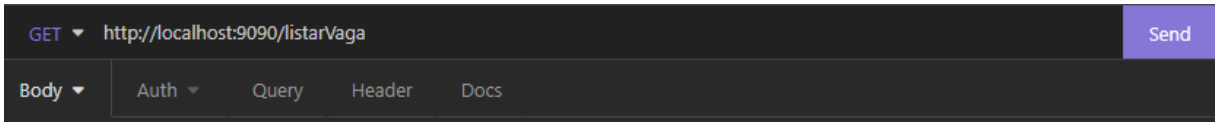

Figura 12. Teste do *endpoint* listarVagas via Insomnia

Assim, ao realizar a requisição, é retornado um *JSON* contendo o erro "*Not Found"*, e o caminho que foi acionado pela *URL,* como é mostrado na Figura 13.

Figura 13. Retorno do erro da requisição via Insomnia do *endpoint* listarVaga

5.5.2. Detalhamento de Vagas

A função de detalhamento de vagas, consiste em fornecer todas as informações referentes a uma vaga, passado um determinado *id* via requisição *HTTP*, como é exemplificado na Figura 5. Seu teste foi primeiramente escrito na *Sprint* 1, e foi desenvolvida e implementada na *Sprint* 3, como mostra a Tabela 3.

5.5.2.1. *Backend*

5.5.2.1.1. Caso de Teste: Cenário Positivo

A Tabela 6 traz a descrição do teste *backend* através da funcionalidade "detalharVaga". Ao realizar uma requisição *HTTP* no *endpoint* /detalharVaga, é passado o parâmetro referente ao id correspondente da vaga que deverá ser detalhada, em formato *JSON* diretamente na *URL*: http://localhost:9090/detalharVaga?vaga={ "id": 4}, assim, identificando a vaga referente ao código desse id para executar a *query* "SELECT NOME, DESCRICAO, CONTEUDO FROM VAGA WHERE ID_VAGA = 4".

Tabela 6. Teste do *endpoint* detalharVaga (*backend*)

Teste	Ferramenta Utilizada	Resultado
Obter os detalhes da vaga correspondente ao id passado	Insomnia	Sucesso

A Figura 14 traz a chamada do *endpoint* "detalharVaga", passando o parâmetro necessário para a busca dos detalhes da vaga referenciada:

Figura 14. Teste do *endpoint* detalharVaga via Insomnia

Se realizada com sucesso, a requisição irá retornar a resposta com um *JSON*, contendo as informações da vaga, como apresenta a Figura 15.

Figura 15. Retorno da requisição via Insomnia do *endpoint* detalharVaga

Como a requisição foi realizada com sucesso, são retornados todos os campos com os detalhes da vaga de acordo com a modelagem do banco de dados na Figura 1.

5.5.2.1.2. Caso de Teste: Cenário Negativo

A Tabela 7 traz a descrição do teste *backend* em um cenário negativo, através da funcionalidade "detalharVaga". Ao realizar uma requisição *HTTP* no *endpoint* /detalharVaga, é passado o parâmetro referente ao *id* correspondente da vaga, só que neste caso de teste sendo um *id* inválido que deverá ocasionar o erro 500 (*Internal Server Error*) na API, e deverá ser retornado no corpo da requisição um *JSON* contendo a causa do erro.

Tabela 6. Teste do *endpoint* detalharVaga (*backend*)

Teste	Ferramenta Utilizada	Resultado
Retornar erro ao tentar obter os detalhes de uma vaga inexistente	Insomnia	Sucesso

A Figura 16 traz a chamada do *endpoint* "detalharVaga", passando o parâmetro necessário gerar o erro no *endpoint*:

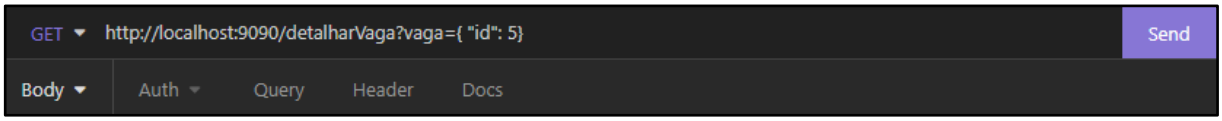

Figura 16. Teste do *endpoint* detalharVaga via Insomnia

A Figura 17 contém o retorno da requisição, no caso um erro 500 (*Internal Server Error*) mostrando a causa do erro, o que era esperado para esse teste.

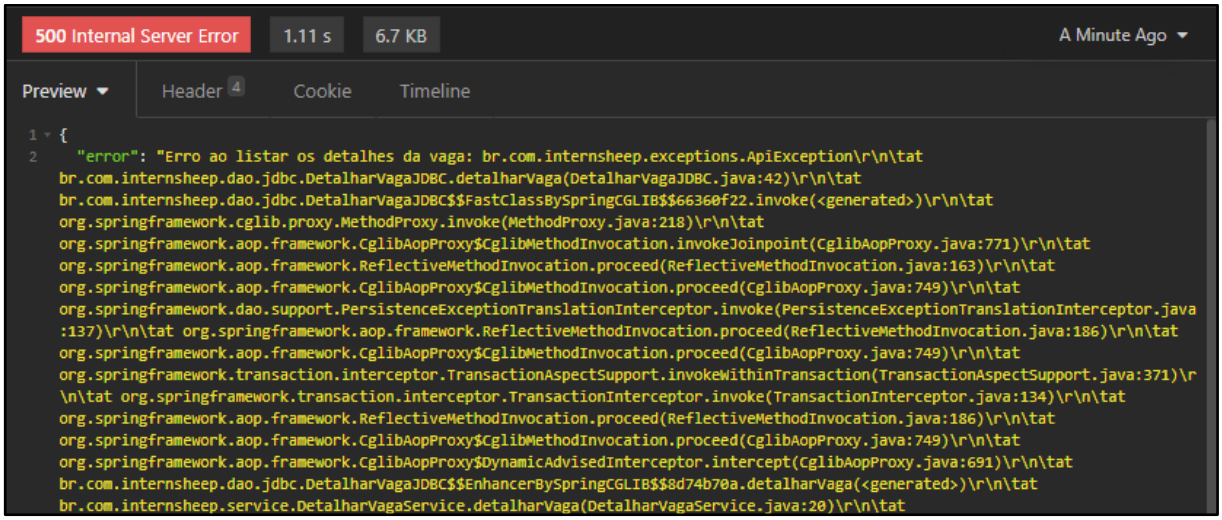

Figura 17. Retorno da requisição via Insomnia do *endpoint* detalharVaga

5.5.3. Cadastrar Vagas

A funcionalidade de cadastrar vaga consiste em inserir uma nova vaga no sistema, de acordo com os parâmetros/informações necessários para a requisição ser completa com sucesso, estando alinhada com os campos que foram definidos na modelagem do banco de dados, como mostrado na Figura 1. Essa funcionalidade foi definida na *Sprint 1* e desenvolvida na *Sprint* 4.

5.5.3.1. *Backend*

5.5.3.1.1. Caso de Teste: Cenário Positivo

A Tabela 7 traz a descrição do teste *backend* através da funcionalidade “cadastrarVaga”. Com o acesso, o *endpoint* /cadastrarVaga, irá criar uma nova vaga no sistema. Em sua requisição *HTTP*, deverão ser passados todos os dados referentes à vaga por *URL*, como por exemplo:

http://localhost:9090/cadastrar?vaga={"nome": "teste","descricao": "teste ", "conteudo": "teste ", "tipoVaga": "estagio"}

Tabela 7. Teste positivo do *endpoint* cadastrarVaga *(backend)*

Teste	Ferramenta Utilizada	Resultado
Cadastrar uma nova vaga com sucesso no sistema	Insomnia	Sucesso

Assim o mesmo irá executar a *query* “INSERT INTO VAGA (NOME, DESCRICAO, CONTEUDO, TIPO_VAGA) VALUES ('teste', 'teste', ‘teste’, ‘estagio’)”, quando o *endpoint* for acionado, passando os valores informados na *url* como parâmetro, como mostra a Figura 18.

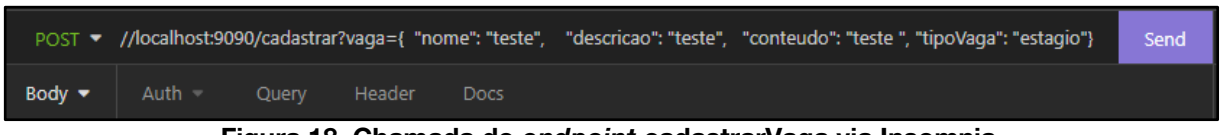

Figura 18. Chamada do *endpoint* cadastrarVaga via Insomnia

Após o envio de requisição, será retornado se a vaga foi criada com sucesso no formato *JSON* como resposta da requisição, como apresentado na Figura 19.

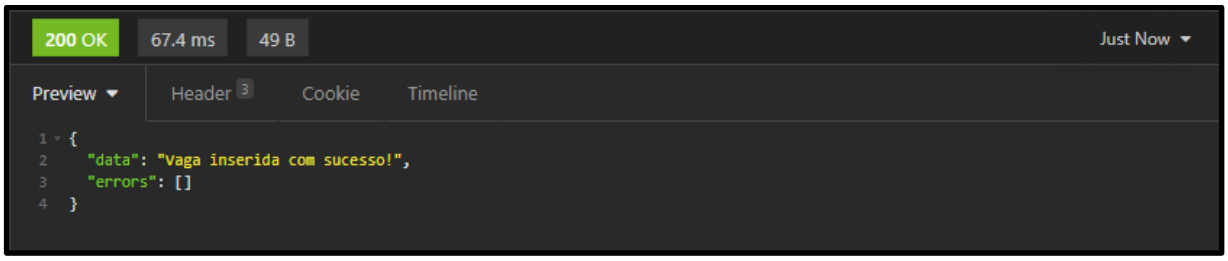

Figura 19. Retorno da requisição via Insomnia do *endpoint* cadastrarVaga

5.5.3.1.2. Caso de Teste: Cenário Negativo

A Tabela 8 traz a descrição do teste *backend* com valores inválidos através da funcionalidade "cadastrarVaga". Com o acesso, o *endpoint* /cadastrarVaga, deverá criar uma nova vaga no sistema, mas caso for passado em sua *URL* algum campo com o nome inválido, o mesmo deverá retornar erro 500 (*Internal Server Error*), e a causa do motivo de seu erro, como mostrado no exemplo a seguir:

http://localhost:9090/cadastrar?vaga={"nme": "teste7", "descricao": "teste7", "conteudo": "teste7", "tipoVaga": "trainee", "linkVaga": "www.teste.com.br"}

Tabela 8. Teste negativo do *endpoint* cadastrarVaga *(backend)*

Teste	Ferramenta Utilizada	Resultado
Forçar erro ao cadastrar nova vaga no sistema	Insomnia	Sucesso

Será executada a mesma *query* mencionada no caso de teste positivo do capítulo 5.5.3.1.1, com os valores passados na requisição, como mostra a Figura 20.

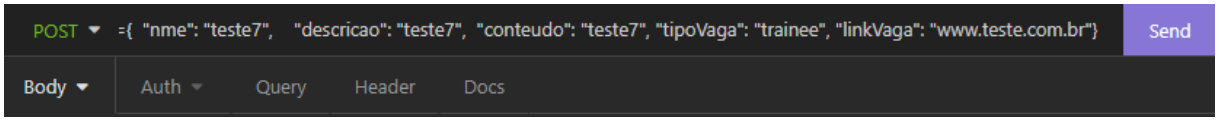

Figura 20. Chamada do *endpoint* cadastrarVaga via Insomnia

Com o envio da requisição feito, a *API* irá retornar o erro que ocorreu durante a requisição no campo "*errors*" no formato JSON, como mostra na Figura 21.

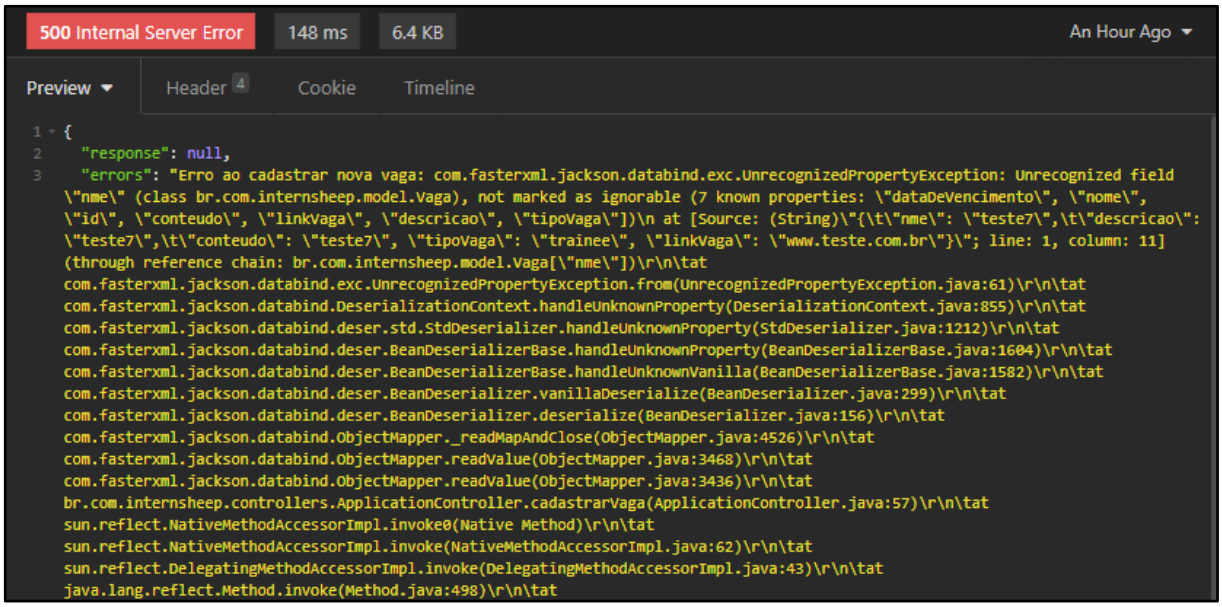

Figura 21. Retorno da requisição via Insomnia do *endpoint*

Nesse caso, a requisição não foi concluída e foi retornado que houve erro interno da aplicação, sendo o erro esperado para esse cenário de teste.

6. Conclusão

O propósito deste artigo é a apresentação e o desenvolvimento de um sistema, que consiste em uma *API* de serviços *backend* para o IFSP oferecendo as funcionalidades de administração de vagas de estágio/trainee para o Câmpus. O enfoque é o desenvolvimento ágil com a metodologia *Scrum* conjuntamente com a escrita de testes de acordo com o que é pregado no *TDD*, e, no que se diz respeito à codificação que teve amplo uso do Spring Framework para o desenvolvimento do código em Java e sua separação de arquitetura em pacotes abordando o uso do conceito de *clean code*.

Ao final do trabalho, teve-se êxito em todos os testes que foram propostos e desenvolvidos durante o levantamento de requisitos e no decorrer das *Sprints* propostas, assim resultando em uma *API* integrada com banco de dados na linguagem MySQL totalmente funcional.

É importante ressaltar que matérias como Banco de Dados I e II para a construção e modelagem do banco de dados do projeto, Engenharia de *Software* para a definição de *Sprints*, dos casos, testes e toda a parte do ciclo de desenvolvimento e Linguagem de Programação III foram imprescindíveis para o desenvolvimento deste trabalho, além da Programação Web quanto aos conceitos da linguagem Java *Web*.

7. Trabalhos Futuros

Propõe-se um estudo de login integrando a autenticação do usuário por meio do SUAP, como é feito atualmente com os alunos que acessam o ambiente virtual Moodle. Esta integração facilitará o acesso através de um cadastro já existente, juntamente com a construção de uma *API frontend* que se integre com a atual produzida, estando apta para mostrar corretamente as informações armazenadas no banco de dados obtidas pela *API backend*.

É proposta também a integração desse sistema com a do autor [Salvino 2019], para que se tenha uma ferramenta final robusta com a divulgação de estágios, e seu gerenciamento.

Referências

ANGULARJS. (2021). “What Is AngularJS?”. https://docs.angularjs.org/guide/introduction. [Online: acessado em 27 de julho de 2021].

ANICHE, M. F. Como a prática TDD influencia o projeto de classes em sistemas orientados a objetos. Orientador: Prof. Dr. Marco Aurélio Gerosa. 2012. Dissertação (Mestre em Ciência da Computação) - Universidade de São Paulo, [S. l.], 2012. Disponível em: https://teses.usp.br/teses/disponiveis/45/45134/tde-31072012-181230/publico/dissertacao.pdf. Acesso em: 25 jul. 2021.

DEVMEDIA. (2021). “Conhecendo o Spring Framework”. https://www.devmedia.com.br/guia/guia-de-spring/37806. [Online: acessado em 27 de julho de 2021].

ELMASRI, R., NAVATHE, S. B. (2018). “Sistemas de Banco de Dados”. Pearson, 7ª Edição.

HEUSER, C. A. (2009). “Projeto de Banco de Dados”. Bookman, 6ª Edição.

SALVINO, L. T. (2019). “Sistema Gerenciador de Estágio do IFSP Hortolândia com arquitetura de microsserviços”. https://hto.ifsp.edu.br/portal/images/thumbnails/images/IFSP/Cursos/Coord_ADS/Arquivo

s/TCCs/2019/TCC_Lenilson_Teixeira_Salvino.pdf. [Online: acessado em 20 de agosto de 2021].

SILVA, M. (2010). “JavaScript. Guia do Programador”. Editora Novatec.

SOARES, M. S. (2004). “Metodologias Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software”. http://periodicosibepes.org.br/index.php/reinfo/article/view/146/38. [Online: acessado em 26 de julho de 2021]..

SMITH, B. (2015). “JSON Básico”. Editora Novatec.

SOMMERVILLE, I. (2011). “Engenharia de Software”. Pearson, 9ª edição.