BRUNO JAHEL MEIRELES

DESENVOLVIMENTO DE APLICAÇÕES WEB DURANTE ESTÁGIO NA DTI

LAVRAS – MG

2020

BRUNO JAHEL MEIRELES

DESENVOLVIMENTO DE APLICAÇÕES WEB DURANTE ESTÁGIO NA DTI

Relatório de estágio supervisionado apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

Prof. Dr. Erick Galani Maziero
Orientador

LAVRAS – MG

2020

Ficha catalográfica elaborada pela Coordenadoria de Processos Técnicos

da Biblioteca Universitária da UFLA

Jahel Meireles, Bruno

Desenvolvimento de aplicações WEB durante estágio na DTI / Bruno Jahel Meireles. – Lavras : UFLA, 2020.

49 p. :

Relatório de estágio (Graduação)–Universidade Federal de Lavras, 2020.

Orientador: Prof. Dr. Erick Galani Maziero.

Bibliografia.

1. Spring Boot. 2. AngularJS. 3. Back-end. 4. Sistemas Web. I. Universidade Federal de Lavras. II. Título.

BRUNO JAHEL MEIRELES

DESENVOLVIMENTO DE APLICAÇÕES WEB DURANTE ESTÁGIO NA DTI

Relatório de estágio supervisionado apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

APROVADA em 04/08/2020.

Prof. Dr. Erick Galani Maziero	UFLA
Prof. Dr. Rafael Serapilha Durelli	UFLA
Prof. Dr. Paulo Afonso Parreira Junior	UFLA
Thiago Gonçalves Cardoso Resende	dti

Prof. Dr. Erick Galani Maziero
Orientador

LAVRAS – MG
2020

Aos meus pais e meu querido irmão, por sempre terem apoiado minhas decisões e me incentivarem a lutar pelos meus sonhos.

AGRADECIMENTOS

Ao professor Erick Maziero, presidente da banca e meu professor orientador, pela ajuda, ensinamentos e suporte durante o curso e a orientação dessa dissertação;

Aos avaliadores da banca, Rafael Durelli, Paulo Afonso Parreira e Thiago Resende, que já me orientaram em outras circunstâncias durante a graduação, pela disponibilidade e paciência;

À UFLA por me proporcionar um ambiente de crescimento pessoal e profissional;

À dti, por me apresentar um ambiente de trabalho onde eu posso me expressar com liberdade e por me apresentar profissionais incríveis que tenho como modelo para a minha vida profissional;

Ao meu amigo Paulo Pedroso, por ser um exemplo de persistência, por ser minha companhia durante a quarentena e por sempre me lembrar que tudo ia dar certo;

Ao meu amigo Harryson Guimarães pela atenção, cuidados e conselhos para que eu me torne um profissional e uma pessoa melhor a cada dia;

À minha amiga Danielle Peixoto por ter me apresentado a UFLA, ter sido uma companheira para várias ocasiões e por ser um exemplo de superação;

Ao meu amigo Guilherme Teixeira por ter sido meu companheiro acadêmico desde o início desta jornada;

À todos que eu tive o prazer de conhecer durante essa trajetória e que me ajudaram a chegar onde estou hoje, obrigado!

Meus filhos terão computadores, sim, mas antes terão livros. Sem livros, sem leitura, os nossos filhos serão incapazes de escrever – inclusive a sua própria história.

Bill Gates

RESUMO

Este relatório de estágio apresenta a trajetória do estudante na empresa dti, as principais atividades realizadas e os conhecimentos adquiridos, relacionando o aprendizado obtido durante a graduação com as tarefas realizadas no dia-a-dia da empresa. Vários frameworks e ferramentas auxiliares são empregados nos projetos da dti, como o Spring Boot e AngularJS. Serão descritas as dificuldades encontradas no desenvolvimento de uma plataforma para contribuição de dados. Também será apresentado um panorama geral da dti e metodologias empregadas nos projetos. O estágio tem como objetivo principal aperfeiçoar as técnicas de programação, bem como adquirir experiência em desenvolvimento de sistemas, para que o estudante se torne um profissional qualificado e pronto para o mercado de trabalho.

Palavras-chave: Spring Boot. AngularJS. Back-end. Sistemas Web.

ABSTRACT

This internship report presents the student's journey in the company dti, the main activities and the obtained knowledge. Also, the gained learning during graduation is discussed in relation to the tasks executed in the company's everyday activities. Many frameworks and auxiliary tools were employed in dti projects, such as Spring Boot and AngularJS. The difficulties encountered in developing a platform for data contribution will be described. An overview of dti and methodologies used in the projects will also be presented. The internship had as main objective the improvement of the programming techniques, as well as the acquisition of experience in system development, in order to allow to the student becomes a qualified professional and ready for the job market.

Keywords: Spring Boot. AngularJS. Back-end. Web Systems.

LISTA DE FIGURAS

LISTA DE SIGLAS

API	*Application Programming Interface*
JVM	*Java Virtual Machine*
XML	*Extensible Markup Language*
API	*Application Programming Interface*
REST	*Representional State Transfer*
HTTP	*Hypertext Transfer Protocol*
JSON	*JavaScript Object Notation*
URI	*Uniform Resource Identifier*
HTML	*HyperText Markup Language*
TCP	*Transmission Control Protocol*
CSS	*Cascading Style Sheets*
SGBD	Sistema de Gerenciamento de Banco de Dados
IDE	*Integrated Development Environment*
RFC	*Request for Comments*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta a dti sistemas, empresa onde o estudante realizou o estágio, bem como o organograma da empresa e as principais funções do estagiário no contexto em que esteve inserido.

1.1 A dti sistemas

A dti foi fundada em novembro de 2009, possui sede em Belo Horizonte/MG com dois escritórios e filiais em Lavras/MG e São Paulo/SP. Também está em processo de criação de uma filial no Canadá. Possui atualmente 654 funcionários e não possui um ramo específico de trabalho na área de desenvolvimento, provendo soluções para qualquer área de negócio. A maioria dos produtos são construídos do zero, desde a concepção.

Os sócios fundadores da dti trabalhavam juntos na Atan, empresa que tinha como foco principal soluções de automação industrial, desenvolvendo sistemas e atuando em toda a pirâmide de automação industrial com soluções sob medida para cada cliente. Em 2001, quando foi lançado o Manifesto Ágil, Marcelo Szuster (um dos sócios fundadores da dti, e nessa época, gerente de TI da Atan) começou a estudá-lo com o objetivo de aplicar as práticas ágeis no departamento de TI. Em 2008, a Atan foi comprada por uma grande multinacional, mudando todas as metodologias e procedimentos utilizadas na empresa. Szuster queria continuar trabalhando com metodologias ágeis e então se juntou a outros colegas de empresa e decidiram fundar a dti.

A estrutura organizacional da dti é dividida em alianças, guildas, tribos e *squads*. Uma aliança é um grupo de tribos, que por sua vez são times que prestam serviço geralmente (mas não obrigatoriamente) para um mesmo cliente. Guildas são uma estrutura paralela às alianças e podem ser definidas como grupos de conhecimento. Já os *squads* são times de profissionais que trabalham num mesmo projeto. A dti preza muito pela descentralização das decisões, pois um dos prin-

cipais pilares da empresa é de que os profissionais devem ter autonomia para trabalhar. Portanto, adota uma gestão de hierarquia horizontal, onde as tribos são auto-suficientes, facilitando a viabilidade de crescimento da empresa em quantidade de tribos. As tribos da dti possuem autonomia sobre gastos, contratações e demissões.

O objetivo principal da dti é se posicionar cada vez mais como um parceiro que ajuda a promover a transformação digital e não apenas desenvolver software para seus clientes. Preza pela relação entre ela e os clientes mesmo quando o software é entregue, buscando manter a parceria. O grande diferencial da dti é priorizar o cliente, disponibilizando uma tribo para que o cliente tenha contato direto com uma equipe exclusiva para os seus projetos.

1.1.1 O projeto

O sistema em que o estagiário trabalhou é uma ferramenta de colaboração de dados. Porém, por questões de sigilo de contrato com a dti, os dados sensíveis do projeto e da empresa não poderão ser citados.

Este sistema tem como premissa inicial disponibilizar uma interface amigável na aplicação Microsoft Excel para que os colaboradores da empresa cliente possam contribuir com informações de interesse da empresa em um banco de dados. As contribuições são enviadas por um suplemento do Excel em formato de coleções, que podem ser definidas como conjuntos de atributos pré-definidos. Estes dados são inseridos numa planilha, coletados através do suplemento, e são enviados para o *backend* através de uma API RESTful desenvolvida em Spring Boot.

Após a coleta de dados, os mesmos servem de insumo para planejamentos e tomadas de decisão dentro da empresa. São disponibilizados para os vários processos internos através de várias interfaces, entre elas a mesma em que foram

inseridos ou através de outros sistemas, como geração de gráficos através de ferramentas de *Business Inteligence*.

1.2 Papel do estagiário

Dentro da dti, o estagiário atuou no detalhamento de requisitos, modelagem de processos e de software, e principalmente desenvolvimento de software. Participou também de vários ritos ágeis promovidos através da cultura da empresa, sendo eles: reuniões diárias para acompanhamento das atividades pelo *squad* e pelo cliente; entrevistas com o cliente para inclusão de novos itens de desenvolvimento; reuniões de repasse de atividades; validações de atividades e revisões de código em dupla; análise de risco das atividades a serem desenvolvidas na *sprint*; e eventuais 'checks de execução' para saber se todos esses ritos estão sendo executados com qualidade.

2 REFERENCIAL TEÓRICO

Serão descritas, neste capítulo, as principais tecnologias e linguagens de programação utilizadas pelo estagiário na dti.

2.1 Java

Java é uma plataforma de desenvolvimento e possui uma linguagem de programação de alto nível orientada a objetos. É implementada a partir de classes, em que o programador define as propriedades (ou atributos) que aquela classe tem e os métodos (funções que idealmente manipulam apenas os atributos do próprio objeto) que ela possui (RICARTE, 2001).

Java foi lançada pela Sun em 1995 e o slogan utilizado para descrever os benefícios da portabilidade da linguagem é "Escreva uma vez, execute em qualquer lugar". Segundo o RedMonk, um site que mede os rankings de popularidade de cada linguagem de programação através de dados do GitHub e discussões no *Stack Overflow*, Java é a segunda linguagem mais popular, mundialmente (O'GRADY, 2020).

Códigos nesta linguagem, que podem ser desenvolvidos em qualquer ambiente, são compilados para um *bytecode*[1] padrão, e podem ser executados em qualquer dispositivo equipado com uma *Java Virtual Machine* (JVM). O site oficial da Oracle, empresa que mantém o Java atualmente, diz que existem hoje 45 bilhões de máquinas virtuais executando Java. Entre os dispositivos que a executam, podemos citar computadores, geladeiras, caixas eletrônicos, celulares, televisores e muito mais.

Uma das várias vantagens de utilizá-la é que a comunidade de programadores Java é muito forte ao redor do mundo. Através dela é possível encontrar

[1] O *bytecode* pode ser entendido como uma linguagem intermediária entre o código-fonte e a aplicação final. Foi projetado focando nas necessidades da linguagem Java, e não dos Sistemas Operacionais em que será executada.

vários materiais de estudo facilmente, participar de encontros, palestras e minicursos. Outra vantagem é a variedade de *frameworks*[2] desenvolvidos em Java. Um dos frameworks mais utilizados é o Spring Boot, que será abordado na próxima seção.

2.2 Spring Boot

Spring Boot é um framework da Spring[3] que simplifica a configuração inicial de aplicações *backend* em Java, favorecendo a convenção sobre a configuração. Basta informar quais módulos deseja utilizar (Template Web, segurança, persistência, entre vários outros) e a configuração será feita automaticamente.

A configuração dos módulos de que a aplicação trabalhada nesta dissertação depende é feita num arquivo chamado *pom* e é escrito na linguagem XML, desenvolvida para facilitar compartilhamento de informações na rede. O arquivo *pom* é utilizado pelo Maven para executar o *build* do projeto e nele estão presentes além das dependências, informações sobre o projeto e detalhes de configuração (MAVEN, 2020). Além de permitir a customização através de módulos, o usuário também pode desenvolver suas próprias configurações.

O maior benefício da utilização do Spring Boot para aplicações *backend* é poder diminuir o tempo de configuração de uma aplicação, facilitando o desenvolvimento e aumentando a produtividade. É muito utilizado para arquiteturas que se baseiam nos conceitos de microsserviços, simplificando o *deploy* de aplicações distribuídas.

Diferente de uma aplicação *frontend*, que lida com as interações diretas na WEB através de elementos visuais, uma aplicação *backend* é responsável pelas

[2] Um *framework* é um conjunto de bibliotecas ou componentes que são utilizados para facilitar a resolução de problemas, comumente encontrados durante a programação. Apresentam uma abordagem genérica, permitindo que o programador dirija seus esforços para a resolução do seu problema em vez de reescrever código.

[3] Spring é um projeto *open-source* desenvolvido pela empresa Pivotal Software e que serviu de base para criação do framework Spring Boot

interações indiretas do usuário, como acesso ao banco de dados, regras de negócio, lógicas de segurança e a API[4] de comunicação com o *frontend*. Um desenvolvedor *backend* é responsável não só pelo desenvolvimento das regras de negócio, mas também pela arquitetura e escalabilidade do sistema.

2.3 API RESTful

Dentro do universo das API's, existe um padrão conhecido como RESTful. Ele é baseado no protocolo HTTP (protocolo de comunicação base da Web), capaz de se comunicar com outras aplicações através da implementação de princípios da arquitetura Web chamada REST. O REST consiste em regras que permitem a criação de um aplicativo com interfaces de comunicação bem definidas, abstraindo os detalhes de implementação dos componentes, focando nos papéis para os quais foram desenvolvidos. Para isso, utilizam as verbos (ou ações) HTTP.

Os verbos HTTP definem qual é a finalidade de uma requisição a que o sistema deve responder. Existem 39 RFCs[5] de especificação de verbos HTTP, dos quais listaremos apenas os utilizados nos projetos em que o estagiário trabalhou:

- **GET:** É utilizado para ler ou recuperar a representação de um recurso específico. Em caso de sucesso, retorna uma representação de um objeto. Deve apenas retornar dados, e nunca modificá-los. Este método é idempotente, ou seja, o resultado será o mesmo caso duas solicitações idênticas sejam realizadas, desde que não haja modificação dos dados por outras entidades entre as requisições.

[4] Uma API, ou Interface de programação de aplicações, em tradução livre, é um conjunto de padrões estabelecidos de um software para a comunicação com outras aplicações através da disponibilização de serviços, chamados de *endpoints*

[5] Um RFC, ou Pedido para Comentários em tradução livre, é um documento técnico desenvolvido pelo IETF - *Internet Engineering Task Force*, uma instituição que especifica os padrões implementados em toda Web. Cada RFC deve detalhar o funcionamento e especificações necessárias para implementar um protocolo.

- **PUT:** O verbo PUT é utilizado para atualizar um recurso do sistema, enviando a requisição para o seu identificador único, ou URI. O corpo dessa requisição contém a versão mais atualizada do recurso. Também é idempotente, pois ao atualizar um recurso e repetir a ação com o mesmo corpo, o recurso ainda estará lá, e contendo a mesma representação de estado.

- **POST:** É frequentemente usado para criar novos recursos. Também é utilizado para atualizar recursos em casos especiais em que a atualização não é idempotente, visto que duas requisições idênticas de POST devem produzir itens diferentes com o mesmo conjunto de dados.

- **OPTIONS:** O cliente, ao solicitar uma requisição do tipo OPTIONS, receberá como resposta quais as opções de requisição (ou quais verbos) para aquele recurso estão disponíveis. Pode ser entendida como uma requisição para obter as informações de comunicação, sem implicar em alguma ação de recuperação de recurso.

- **DELETE:** Como o próprio nome diz, este verbo é utilizado para excluir um recurso identificado pelo seu URI. É idempotente pois tentar deletar o mesmo recurso duas vezes, gera o mesmo estado final: inexistência do recurso.

2.4 HTTP

Todas as trocas de dados na Web são baseadas em HTTP, que é um protocolo de comunicação do tipo cliente-servidor, permitindo a obtenção de recursos, como documentos HTML. Este tipo de comunicação é iniciado sempre pelo destinatário, que é geralmente - mas não unicamente - um navegador, chamado de

user-agent[6] (FIELDING et al., 1999). Clientes e servidores se comunicam através de mensagens chamadas de requisições e respostas, respectivamente.

Este protocolo da camada de aplicação é enviado sobre o protocolo de rede TCP, garantindo confiabilidade ao HTTP, visto que o recebimento e confirmação de pacotes são assegurados. É caracterizado por possuir requisições independentes ou *stateless*, que não guardam informações entre elas. Isso faz com que as requisições REST sejam menos complexas, já que não possuem lógica de sincronização de estados. Ser *stateless* facilita a utilização de cache.

Cache é uma ferramenta de memória interna de dados que age como um intermediário entre o cliente e o servidor, aumentando a velocidade de navegação de dados a partir de acessos futuros. Consegue fazer isso armazenando a resposta 'mais perto' do cliente, de forma que este receba a resposta mais rápido, e, como consequência, diminui o congestionamento da rede.

2.5 Microsserviços

Os microsserviços são uma abordagem estrutural de construção de software que consiste na quebra das funcionalidades de uma aplicação em pequenos módulos, de forma que cada módulo funciona como um sistema único, disponibilizando seus serviços através de uma API. Esse estilo de desenvolvimento permite que cada módulo tenha sua própria linha de produção, operações e versionamento.

Diferente dos sistemas monolíticos, onde qualquer mínima mudança precisa que todo o sistema passe por um *build* e *deploy*, um sistema que é desenvolvido em microsserviços pode realizar alterações pontuais em cada um dos seus módulos, otimizando o tempo para implantar funcionalidades novas e sem que toda a aplicação fique *offline* (FOWLER, 2014).

[6] O *user-agent* é qualquer ferramenta que age em nome do usuário. É predominantemente realizada por navegadores Web ou programas depuradores de aplicações Web

Várias vantagens são frequentemente associadas à eles, as principais são a velocidade, autonomia, escalabilidade e independência (JAMSHIDI et al., 2018). Desde que sua interface de comunicação esteja bem estruturada, cada módulo pode ser implementado em diferentes linguagens de programação e até por times diferentes. Caso a demanda de um único microsserviço seja alta, é possível aumentar a quantidade de instâncias apenas daquele módulo, sem impactos para os consumidores do microsserviço (XIAO; WIJEGUNARATNE; QIANG, 2016).

Utilizando essa estrutura, grandes aplicações podem ser separadas em pequenos serviços, facilitando correções, novas funcionalidades e replicação sob demanda.

2.6 JUnit

JUnit é um *framework open-source* para escrita de testes unitários para código desenvolvido na linguagem Java. Teste unitário é uma modalidade de testes que é focada na verificação da menor unidade de um software (um método ou classe). Com o teste de cada unidade básica e suas integrações, ao ocorrer algum erro é possível apontar exatamente qual parte está com problemas, otimizando a resolução de *bugs*.

O JUnit possui uma interface completa para construir e executar testes, disponibilizando várias funções úteis para verificar funcionalidades do código. Permite também a automação de execução de todos os testes antes do *deploy* da aplicação, garantindo a integridade e estabilidade do *software*. Automatizando também alguns detalhes da escrita de código unitário, faz os testes serem mais fáceis de construir e permite que sejam construídos de forma incremental (CHEON; LEAVENS, 2002).

2.7 Arquitetura MVC

A arquitetura de um *software* define quais são os elementos e estruturas do mesmo. A arquitetura MVC é um padrão de projeto que separa uma aplicação em três camadas: Modelo, Visão e Controle. É muito utilizada em sistemas Web atualmente, e tem como principal conceito a separação de partes distintas do projeto, reduzindo ao máximo suas dependências.

Na camada de Modelo, são realizadas definições e manipulações sobre os dados. É responsável pela leitura, registro e validação deles. Já na camada de Visão encontramos toda a interface gráfica com que o usuário tem contato. A última camada deste padrão, o Controle, é responsável por intermediar as requisições do usuário, direcionando as suas solicitações para a execução das regras de negócio e retornando os resultados para a camada de Visão.

A utilização do padrão MVC isola as regras de negócios, da lógica da interface, possibilitando a criação de várias interfaces sem que haja alteração das regras de negócio. Isso traz flexibilidade e possibilidade de reutilização de classes, economizando esforço para disponibilizar os mesmos dados em interfaces diferentes.

2.8 Angular.JS

Angular.JS é um *framework* estrutural desenvolvido pelo Google para páginas web dinâmicas. Permite a extensão da sintaxe do HTML, para adição de novas funcionalidades como *data binding* e injeção de dependências através da manipulação do DOM[7], diminuindo substancialmente a quantidade de código HTML necessário para a criação de uma interface. É ideal para aplicações CRUD[8] e já não

[7] O DOM (*Document Object Model*) é uma interface que representa como os documentos HTML e XML são lidos pelo browser. É uma representação estruturada da página HTML e define meios de como essa estrutura pode ser acessada.

[8] Aplicações CRUD são aplicações que permitem realizar quatro operações básicas sobre dados: *Create*, *Read*, *Update* e *Delete* (Criar, Ler, Atualizar e Deletar).

é tão indicada para aplicações como jogos ou editores gráficos, pela complexidade das manipulações do DOM necessárias.

2.9 Suplementos Excel

Os suplementos foram criados para estender as funcionalidades do Excel, sendo portáveis para várias plataformas, como sistemas Windows e Mac. São desenvolvidos em HTML, CSS e JavaScript utilizando uma API especial para o pacote Microsoft Office. Além de permitir a interação com os dados da planilha, facilita a interação com recursos online, podendo enviar requisições e executar autorizações OAuth 2.0[9]. Também possuem suporte para *frameworks*, como o Angular.JS e jQuery.

Uma das suas principais vantagens é a possibilidade de trazer uma página Web com todas as funcionalidades de um sistema *frontend* padrão para o ambiente Office, e esta pode interagir com a planilha, lendo ou gravando dados. Outra vantagem é a facilidade de implantação, pois o aplicativo Web pode ser hospedado em qualquer servidor e distribuído através da loja de suplementos do Office. Na loja podemos encontrar todo tipo de suplementos, como criação de faturas Paypal, inserção de mapas do Google Maps, dicionários e até mesmo pesquisas com resultados exportados diretamente para a planilha.

2.10 GIT

Git é um sistema *open-source* poderoso e flexível de versionamento de código desenvolvido em 2005 por Linus Torvalds. Linus decidiu criá-lo por não existirem na época sistemas de versionamento fáceis de escalar, que pudessem facilitar o desenvolvimento distribuído de forma eficiente (LOELIGER; MCCULLOUGH, 2012).

[9] OAuth é um protocolo de autorização entre aplicações que utiliza tokens de acesso. Permite a autenticação segura, sem expor as credenciais (*login* e senha) do usuário.

É utilizado atualmente pela grande maioria dos desenvolvedores e empresas. Ajuda a controlar o fluxo de novas funcionalidades de um sistema e dispõe de características que permitem que vários usuários contribuam para o desenvolvimento simultaneamente. Isso é feito através de *branchs*, que são versões paralelas de um mesmo arquivo num repositório (BLISCHAK; DAVENPORT; WILSON, 2016). Possui integração com várias IDEs, permitindo que seja utilizado sem sair da interface de desenvolvimento.

2.11 SCRUM

Scrum é uma metodologia extremamente ágil e flexível, que tem por objetivo definir um processo de desenvolvimento interativo e incremental, podendo ser aplicado a qualquer produto ou gerenciamento de qualquer atividade complexa (BISSI, 2007). Baseia-se no desenvolvimento incremental das aplicações centrado na equipe, com ciclos de iteração curto, chamados de *sprints*. Ou seja, o Scrum não é uma técnica para construção de produtos, é um conjunto de ritos, processos e métodos que auxiliam no rápido desenvolvimento de um produto ou tarefa, visando o aumento da comunicação e cooperação entre os membros da equipe.

Ao final de cada *sprint*, é desejável que seja entregue algo que agregue valor ao produto, permitindo que o cliente faça pequenas validações a cada *sprint*, facilitando a mudança de algo que não esteja do seu agrado, ou que tenha sido desenvolvido de forma discrepante do que foi acordado anteriormente. A metodologia possui alguns elementos de apoio, que são:

- ***Backlog* do produto:** Lista de todas as funcionalidades que o produto deve possuir, mas ainda não foram desenvolvidas.

- ***Backlog* da *sprint*:** Lista priorizada pelo cliente, obtida a partir da quebra das atividades do *Backlog* do produto em tarefas menores. É um planejamento do que será desenvolvido durante a *sprint*.

- **Gráficos de acompanhamento:** Gráficos que medem a quantidade de trabalho restante, como o *Burndown*. É recomendado ter um para o produto como um todo, e um para cada *sprint*.

Possui também papéis e responsabilidades, adaptáveis a cada empresa ou projeto. Na dti, temos os seguintes:

- **Cliente:** Participa das tarefas relacionadas à definição da lista de funcionalidades, elaborando requisitos e definindo prioridades das atividades, bem como dos ritos finais de cada *sprint* para validação dos resultados.

- **Desenvolvedor Líder:** Cada *squad* possui um, e geralmente é um desenvolvedor com mais experiência, um ponto de apoio onde os outros desenvolvedores possam tirar suas dúvidas. É encarregado de agendar os ritos do Scrum e gerenciar os trabalhos do *squad*.

- ***Squad*:** Equipe de projeto que possui autoridade de se organizar para atingir os objetivos previstos no documento de requisitos. Possui a responsabilidade de estimar o esforço das atividades, criação e revisão de lista de funcionalidades do produto, e definição de obstáculos que impeçam o desenvolvimento das atividades.

- **Product Owner:** É a ponte entre o time de desenvolvimento e o cliente. Precisa entender bem das regras de negócio do produto, e é o responsável por traduzir as demandas do cliente em uma linguagem que seja mais fácil ao entendimento do *Squad*.

- **Scrum Master:** É o responsável por garantir que o projeto esteja sendo conduzido de acordo com as práticas, ritos e regras do Scrum, removendo quaisquer obstáculos que possam impedir o desenvolvimento do projeto pelo *Squad*, e garantindo que a equipe trabalhe da forma mais produtiva possível.

Alguns dos ritos do Scrum que são realizados dentro do contexto da dti:

- **Planning:** É uma reunião onde os integrantes do *Squad* se reúnem com o Desenvolvedor Líder para definir quais serão as próximas atividades a serem desenvolvidas, estimando a complexidade de realização de cada uma delas.

- **Daily Meeting:** Uma reunião diária, que não deve ultrapassar 15 minutos de duração. Nela é discutido o que foi feito naquele dia, o que cada membro está fazendo nesse momento, qual atividade está prevista para o próximo dia e se existe algum impedimento. Essa reunião é importante pois deixa todo o *Squad* e o Desenvolvedor Líder ciente da atual situação do projeto, e deve ser realizada com todos os presentes de pé, para que seja rápida e outros assuntos não sejam discutidos no momento.

- **Retrospective:** Realizada no final da *sprint*, onde os integrantes do *Squad* relatam pontos positivos, negativos e preocupantes sobre os acontecimentos da *sprint*. Também são discutidas ações para que os pontos negativos sejam amenizados ou sanados.

- **Review:** Também é realizada ao final de cada *sprint*, onde o *Squad* apresenta para o cliente os resultados gerados durante a *sprint* em questão. O cliente então verifica se os itens desenvolvidos estão de acordo com as especificações.

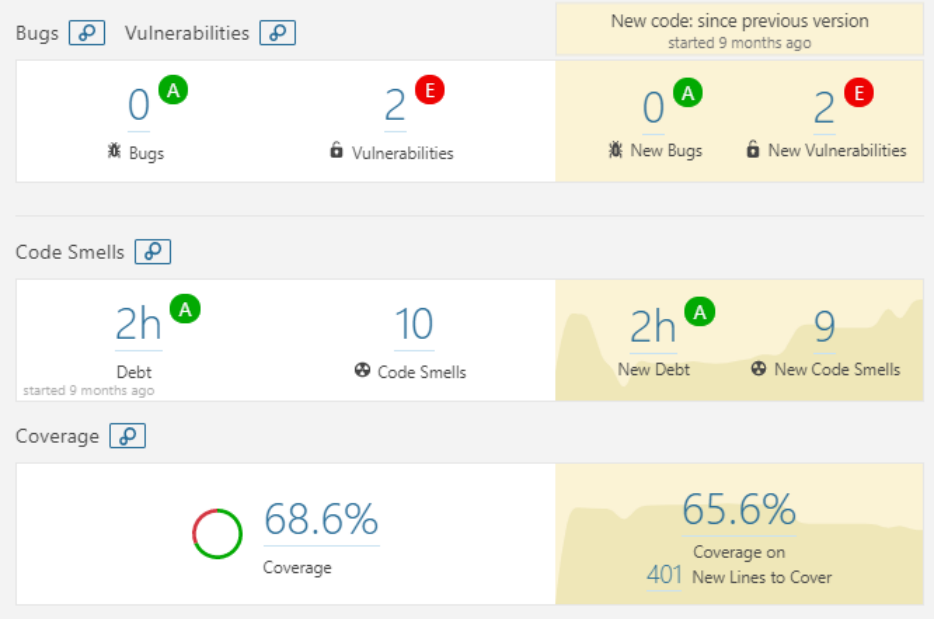
2.12 SonarQube

O SonarQube é uma ferramenta de revisão de código que tem como objetivo localizar *bugs*, vulnerabilidades e *code smells*[10]. Faz isso analisando o código fonte e comparando com centenas de regras de más condutas na sua base, gerando como produto final vários relatórios que podem ser analisados numa linha

[10] *Code Smell* é um termo que caracteriza um trecho de código. A sua tradução livre quer dizer mal cheiro (*Bad Smell*), evidenciando que algo naquele trecho não está de acordo com boas práticas de programação, como códigos duplicados e métodos extensos demais.

do tempo (ARAPIDIS, 2012). Essa função é uma ótima forma de analisar a qualidade do código através do tempo, gerando novos relatórios numa plataforma a cada *deploy*, como pode ser observado na Figura 2.1.

Figura 2.1 – Exemplo de um relatório do SonarQube.



Fonte: dti (2020).

Uma das métricas utilizadas é a de cobertura do código. Ela diz qual a porcentagem do código foi executada pelos testes unitários. Uma baixa cobertura de código revela que os testes unitários não testam a sua aplicação por completo, levando à baixa confiabilidade. É importante salientar que o SonarQube não executa os testes unitários desenvolvidos para o projeto, mas sim utiliza o relatório de execução de outra ferramenta, como o JUnit.

2.13 Microsoft Azure

A Plataforma Azure é um conjunto de serviços disponibilizado pela Microsoft, que provê uma plataforma para execução de aplicações e armazenamento na nuvem. Os clientes desta plataforma têm à sua disposição várias ferramen-

tas como múltiplas instâncias da aplicação, balanceamento de cargas entre as instâncias, armazenamento de dados, rede de distribuição de conteúdo, entre outras (CHAPPELL et al., 2009).

Dentre as múltiplas funcionalidades do Microsoft Azure, as que o estagiário trabalhou ativamente na dti são:

- ***Application Insights*:** Uma plataforma que permite o monitoramento de performance das aplicações, e é altamente customizável. *Logs* de erro são disponibilizados com seu *stack trace*[11], facilitando o rastreamento de exceções. Também gera gráficos de acesso ao longo do tempo com algumas informações dos clientes da aplicação (como o sistema operacional utilizado e sua localização por exemplo), e dos recursos acessados por eles.

- ***Boards*:** Uma seção que provê ferramentas de auxílio para trabalhar com métodos ágeis, como o SCRUM. Possui um quadro onde as atividades em desenvolvimento podem ser categorizadas como 'a fazer', 'fazendo' e 'concluídas'. Também possui funcionalidades de apoio à *sprints*, *backlog* e estimativa de pontos por atividade.

- ***Repo*:** O repositório onde o código da aplicação é armazenado. É uma plataforma que utiliza o GIT, assim como o GitHub ou GitLab. Possui as funcionalidades de um repositório como qualquer outro (*branchs*, *pull requests*, *tags* entre outros).

- ***Pipelines*:** É onde o *build* da aplicação é executado, bem como o seu *deploy*. É aqui que são configurados os ambientes das aplicações (Desenvolvimento, Teste e Produção), sendo possível realizar integração com outras soluções, como o Sonar.

[11] *Stack trace* é a pilha de execução dos métodos de uma aplicação quando um erro não esperado acontece.

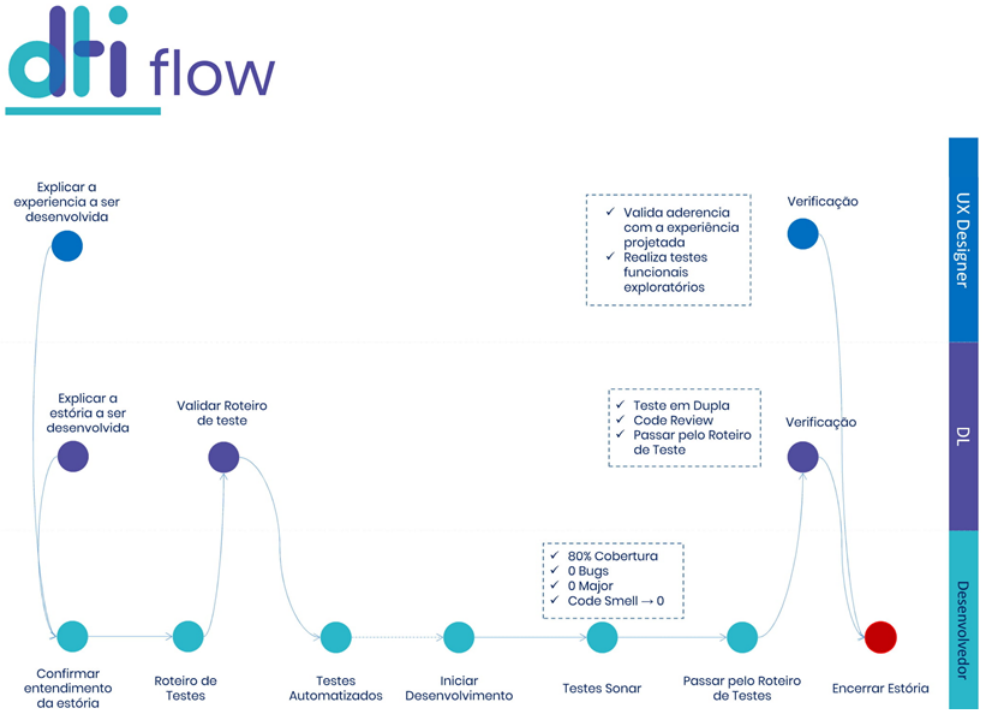
3 METODOLOGIA

Por questões de sigilo, nenhum código ou regra de negócio pôde ser exposta neste trabalho. O projeto possui duas APIs RESTful como *backend*, que provêm *endpoints* de acesso para gerenciar as entidades da aplicação, um microsserviço que também é uma API RESTful e uma interface *frontend* em AngularJS que é executada no Microsoft Excel.

3.1 O Fluxo de uma atividade

Na dti, todas as atividades devem seguir um fluxo interno chamado de DTI Flow, como descrito na Figura 3.1. Ele possui várias etapas que devem ser seguidas para que a atividade realizada seja concluída com qualidade.

Figura 3.1 – Descrição do dti Flow.



Fonte: dti (2020).

Para que uma atividade seja desenvolvida, é necessário que o desenvolvedor faça repasses de entendimento com o Desenvolvedor Líder e com o Designer. Isso assegura que a experiência final para o usuário esteja bem definida. Na dti não existe o papel de *Quality Assurance* (QA) ou comumente conhecido como Tester. O próprio desenvolvedor produz os roteiros de teste, e os valida em dupla com outro integrante do *squad*. Este roteiro deve englobar os casos de sucesso e erro que podem ser encontrados durante o uso da nova funcionalidade. Para que a atividade seja desenvolvida, também é necessária a construção de testes automatizados de forma que haja segurança contra erros, e esses testes também servem de métricas para a cobertura no SonarQube. Só depois dessas etapas o desenvolvimento pode começar.

Após o término do desenvolvimento, o próprio desenvolvedor executa o conjunto de testes que escreveu. A dti fornece um *template* que as equipes podem adaptar de acordo com o seu próprio contexto, como pode ser observado no exemplo da Figura 3.2.

Figura 3.2 – Exemplo de um caso de teste.

Status Geral dos Fluxos de Teste			OK		
Fluxo	**Testes de contribuição da funcionalidade X**		**Status do Fluxo**	**OK**	
1	**Descrição**	**Resultado Esperado**	**Observações**	**Status Developer**	**Status Final**
	Chamada ao serviço de contribuição sem as melhorias de tempo	Armazenar valor de tempo consumido para registro das contribuições		OK	OK
	Chamada ao serviço de contribuição com as melhorias de tempo	Armazenar valor de tempo consumido para registro das contribuições		OK	OK
	Comparação das 2 chamadas anteriores	Valor esperado de redução de tempo em 50%		OK	OK

Fonte: Arquivo Pessoal (2020).

O próximo passo é convidar outro desenvolvedor do time para efetuar os mesmos testes em dupla, caso todos os casos sejam concluídos com sucesso, o *Pull Request* (PR) no Portal Azure é solicitado. No PR é feita a revisão de código.

O último passo, para que a estória seja encerrada, é a validação pelo Desenvolvedor Líder e o Designer.

3.2 Atividades desenvolvidas

Para facilitar a leitura deste relatório, evitando uma lista muito exaustiva de atividades desenvolvidas, as mesmas foram agrupadas em categorias. Elas foram realizadas nas API's RESTful e suplemento do Microsoft Excel já citadas acima, em um banco de dados SQL Server e em ferramentas de DevOps no Portal Azure. Todas as atividades executadas estão inseridas no mesmo contexto e contribuíram para a evolução do projeto.

- **Treinamento:** Na primeira *sprint*, as atividades foram relacionadas à configuração dos ambientes de desenvolvimento e *pair-programming*, que consiste no acompanhamento e desenvolvimento em dupla para que o estagiário conseguisse construir conhecimento sobre o sistema.

- **Banco de Dados:** Foram realizadas atividades no SQL Server, como a construção de *queries* para recuperação de entidades e criação de *Stored Procedures* com o objetivo de transferir dados obsoletos para tabelas secundárias. Esta atividade surgiu devido à quantidade de dados armazenados. O tempo gasto para recuperação dos dados dessas tabelas era muito grande.

- **Portal Azure:** As atividades deste grupo consistiram na configuração de *deploys* automatizados, configuração de *pipelines* e ferramentas auxiliares como o SonarQube, inserção de dados para insumo do *Burndown*, descrição de atividades durante reuniões de *planning* e validação de código em dupla nos *Pull Requests*.

- **Desenvolvimento backend:** Foram realizadas atividades referentes à adição de novos *endpoints* de integração de dados, validação de novos dados

oriundos do suplemento no Microsoft Excel, criação de *features* para entrada de novos dados e integração com banco de dados.

- **Suplemento Microsoft Excel:** Algumas validações dos dados do sistema em que o estagiário trabalhou são feitas no Microsoft Excel, antes mesmo do envio para o *endpoint*. Foram realizadas atividades para construção de novas validações e ajustes de visualização de datas, ajustes de tradução para apoio da internacionalização dos processos e validações de números exponenciais.

- ***Discovery*:** Por ser um sistema legado de outra empresa, o *Squad* não participou da criação ou modelagem inicial do sistema. Portanto, quando o cliente solicitava alterações em funcionalidades nas quais os integrantes nunca haviam trabalhado, existiam atividades em *sprints* anteriores para que houvesse análise do código, de forma que pudéssemos estimar a complexidade destas atividades de forma mais assertiva.

- **Análise de desempenho:** A plataforma de contribuição é um sistema que lida com grande quantidade de dados e portanto deve conseguir realizar suas funções com um desempenho satisfatório. Houve atividades em que era necessário analisar o tempo de requisições, para priorizar quais rotas deveriam ter melhorias, garantido ao usuário final uma experiência mais fluida.

O estagiário já tinha trabalhado com metodologias ágeis antes de iniciar o estágio na dti, portanto já tinha uma certa prática com a metodologia, conseguindo desenvolver as atividades de forma satisfatória. O próximo capítulo trata do encerramento das atividades.

4 CONCLUSÕES

Assim como as disciplinas da graduação serviram de base para o estágio, o estágio serve de base para o mercado de trabalho, numa linha contínua de aprendizado. Esta seção trata da conexão entre as atividades realizadas no estágio com o aprendizado obtido nas disciplinas da graduação. Faz-se, também, uma análise crítica sobre o curso.

4.1 O curso de Ciência da Computação

O aprendizado obtido durante o curso foi imprescindível para o bom desempenho do estagiário em sua atuação na dti. Como as atividades contemplaram várias áreas da computação, a variedade das disciplinas presentes na ementa do curso cobriram grande parte dos conhecimentos necessários para a boa execução das atividades.

O estagiário possui preferência pela área de desenvolvimento, as disciplinas que mais contribuíram para o conhecimento nessa área foram: Introdução aos Algoritmos, Estruturas de Dados, Paradigmas de Linguagens de Programação, Modelagem e Implementação de Software, Algoritmos em Grafos, Práticas de Programação Orientada a Objetos e Sistemas Distribuídos. Através do conhecimento obtido nessas disciplinas, foi possível construir a expertise necessária para trabalhar na solução reportada nesta dissertação. As disciplinas supracitadas contribuíram com conhecimentos que incluem, mas não se limitam a: lógica de programação básica, estruturação de classes e métodos, resolução de problemas, e interação entre sistemas web.

Ao desenvolver software, é importante entender como funciona a sua teoria, arquitetura e estrutura básica, pois é possível modelar o software de forma que ele tenha melhor desempenho de acordo com a sua finalidade e onde será executado. As disciplinas que contribuíram para este tipo de conhecimento foram: Arquitetura de Computadores 1 e 2, Teoria da Computação, Programação Paralela e

Concorrente, Sistemas Operacionais, Sistemas Distribuídos e Compiladores. Com o conhecimento assimilado nessas disciplinas, foi possível ter outra visão de como o software funciona e como a sua interação de forma correta com o hardware pode influenciar na sua velocidade.

Não somente de desenvolvimento se consiste o ambiente para construção de um software. Existem metodologias que auxiliam no desenvolvimento e diferentes processos que concedem agilidade aos projetos. As disciplinas que contribuíram para entender e otimizar este processo complexo foram: Processos de Software, Engenharia de Software, Gerência de Projetos de Software e Inteligência de Negócios. Nestas disciplinas foi possível conhecer um pouco sobre a gama de soluções para construir um software desde a ideia de criação, passando pela documentação de requisitos, projetos arquitetural e de interface, a implementação, validação, entrega e manutenção contínua.

Para que o *software* possa consumir ou prover dados para um banco ou outras aplicações, foi necessário entender como ele se relaciona com estas ferramentas. As disciplinas que auxiliaram nesta área de conhecimento foram: Introdução a Sistemas de Banco de Dados, Sistemas Gerenciadores de Banco de Dados, Redes de Computadores e Sistemas Distribuídos. Com o conhecimento obtido nessas disciplinas foi possível aprender como bancos de dados funcionam, formas de indexação de dados, como os dados tramitam do banco para a aplicação através da rede, comunicação entre aplicações e como funcionam as camadas de rede.

Grande parte do software é desenvolvido para seres humanos, logo, não devemos esquecer deste importante agente na cadeia de conhecimento. As seguintes disciplinas tratavam deste assunto: Interação Humano-Computador, Ética, Computador e Sociedade e Cálculo Numérico. Nessas disciplinas foi possível entender como a relação entre seres humanos e software é uma via de mão dupla. Da mesma forma que os humanos influenciam no software de acordo com suas necessidades, o software também influencia no modo que as pessoas interagem com as

outras e com o mundo. Como os seres humanos são passíveis de erros, também foi necessário estudar como estes erros impactam na precisão dos resultados e a melhor forma de atenuá-los através do arredondamento de casas decimais.

Uma área da computação até então desconhecida pelo estudante foi Aprendizado de Máquina. Apesar de nunca ter trabalhado na área, foi uma experiência incrível descobrir como o software pode realizar simulações do mundo real, aprender a partir de uma amostra de dados e ajudar na tomada de decisão. As disciplinas que trataram desse assunto foram: Mineração de Dados e Inteligência Artificial. Os trabalhos práticos realizados nessas matérias abriram portas para uma ampla visão sobre como podemos utilizar dados para inferir informações sobre uma amostra representativa e como simulações podem oferecer soluções otimizadas.

4.2 Análise crítica

Devido à grande quantidade de áreas que o egresso do curso de Ciência da Computação pode seguir, é inviável ter disciplinas obrigatórias ou eletivas que contemplem todas esses ramos. No caso particular deste estudante que decidiu seguir na área de desenvolvimento WEB, existem matérias eletivas que tratam do assunto. Porém, na área de jogos por exemplo, não existem matérias na grade curricular.

É até difícil escrever sobre todas as áreas de atuação e a falta de disciplinas disponíveis para a graduação, visto que não existe uma disciplina que mostra quais são as áreas existentes e qual o caminho devemos seguir para que possamos chegar até elas.

O corpo docente sempre foi muito solícito, de extrema competência e disposto a ajudar quando necessário. Os docentes com quais o estudante realizou disciplinas, mesmo após o término das mesmas sempre se dispuseram a tirar dúvidas sobre projetos ou trabalhos que o estudante realizou e que tinham como assunto as disciplinas ministradas pelo docente em questão.

4.3 Estágio

Um importante elemento para que o estudante não seja um egresso despreparado para o mercado de trabalho, caso tenha interesse de seguir nesta área, é o estágio. A UFLA dispõe de empresas parceiras como a FUNDECC, que foi onde o estudante realizou o seu primeiro estágio, também em desenvolvimento WEB.

Na dti foi possível exercitar ainda mais os conhecimentos adquiridos, e somar mais conhecimentos relativos à área de desenvolvimento WEB. Por fim, o estudante tem a consciência que o estágio foi essencial para se tornar um bom profissional.

4.4 Dificuldades encontradas

A plataforma de contribuição de dados veio de outra empresa e já estava implantada na empresa cliente. Nem todos os desenvolvedores da dti participaram do projeto inicial do sistema, portanto tiveram que aprender como funcionam as funções da aplicação. Uma das maiores dificuldades em trabalhar neste projeto foi a falta de experiência com o código, pois o mesmo não foi construído com boas práticas de programação. Não sabíamos onde as funcionalidades foram implementadas e o código não era intuitivo. O Sonar não era utilizado no projeto, portanto existiam vários *code smells*, bugs e vulnerabilidades no código, abaixando a sua qualidade. O banco de dados da aplicação é muito complexo e com restrições de alteração pela empresa cliente, sendo possível apenas através de processos burocráticos. Todos esses fatores contribuem para que haja uma queda na velocidade em que as atividades eram desenvolvidas.

4.5 Considerações finais

Num panorama geral, a visão que o estudante tinha do curso era bem limitada ao ingressar na universidade. Durante a graduação, foram apresentados

conhecimentos em diversas áreas da computação que foram fundamentais para o desenvolvimento intelectual, pessoal e profissional. Estes conhecimentos ajudaram no desenvolvimento das atividades durante o estágio, não somente na questão técnica mas também profissional e inter-pessoal, preparando o estudante para que tenha um bom desempenho após o término da graduação.

REFERÊNCIAS

ARAPIDIS, C. **Sonar Code Quality Testing Essentials**. [S.l.]: Packt Publishing Ltd, 2012.

BISSI, W. Metodologia de desenvolvimento ágil. **Campo Digital**, v. 2, n. 1, 2007.

BLISCHAK, J. D.; DAVENPORT, E. R.; WILSON, G. A quick introduction to version control with git and github. **PLoS computational biology**, Public Library of Science San Francisco, CA USA, v. 12, n. 1, p. e1004668, 2016.

CHAPPELL, D. et al. Introducing windows azure. **Microsoft, Dec**, 2009.

CHEON, Y.; LEAVENS, G. T. A simple and practical approach to unit testing: The jml and junit way. In: SPRINGER. **European Conference on Object-Oriented Programming**. [S.l.], 2002. p. 231–255.

FIELDING, R. et al. **Hypertext transfer protocol–HTTP/1.1**. [S.l.]: RFC 2616, june, 1999.

FOWLER, M. **Microservices**. 2014. Acessado em 19/08/2020. Disponível em: <https://martinfowler.com/articles/microservices.html>.

JAMSHIDI, P. et al. Microservices: The journey so far and challenges ahead. **IEEE Software**, IEEE, v. 35, n. 3, p. 24–35, 2018.

LOELIGER, J.; MCCULLOUGH, M. **Version Control with Git: Powerful tools and techniques for collaborative software development**. [S.l.]: "O'Reilly Media, Inc.", 2012.

MAVEN. **Introduction to POM**. 2020. Acessado em 20/08/2020. Disponível em: <https://maven.apache.org/guides/introduction/introduction-to-the-pom.html>.

O'GRADY, S. **The RedMonk Programming Language Rankings: January 2020**. 2020. Disponível em: <https://redmonk.com/sogrady/2020/02/28/language-rankings-1-20/>.

RICARTE, I. L. M. **Programação Orientada a Objetos: uma abordagem com Java**. 2001. 2014 p. Acessado em 02/08/2020. Disponível em: <http://www.dca.fee.unicamp.br/cursos/PooJava/Aulas/poojava.pdf>.

XIAO, Z.; WIJEGUNARATNE, I.; QIANG, X. Reflections on soa and microservices. In: IEEE. **2016 4th International Conference on Enterprise Systems (ES)**. [S.l.], 2016. p. 60–67.