UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS CORNÉLIO PROCÓPIO
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM ENGENHARIA DE SOFTWARE

RODRIGO ROLIM VERAS

SISTEMA WEB PARA GERENCIAR A PONTUAÇÃO DO ALUNO NA DISCIPLINA DE ATIVIDADES COMPLEMENTARES

TRABALHO DE CONCLUSÃO DE CURSO

CORNÉLIO PROCÓPIO

2018

RODRIGO ROLIM VERAS

SISTEMA WEB PARA GERENCIAR A PONTUAÇÃO DO ALUNO NA DISCIPLINA DE ATIVIDADES COMPLEMENTARES

Trabalho de Conclusão de Curso apresentada à disciplina de Trabalho de Conclusão de Curso 1, da Universidade Tecnológica Federal do Paraná, como requisito parcial para obtenção do grau de bacharel em Engenharia de Software

Orientador: Profa Ma. Flávia Blum Haddad

CORNÉLIO PROCÓPIO

2018

Qualquer um pode escrever um código que o computador entenda. Bons programadores escrevem códigos que os humanos entendam.

Martin Fowler

RESUMO

ROLIM, Rodrigo. Sistema web para gerenciar a pontuação do aluno na disciplina de Atividades Complementares. 33 f. Trabalho de Conclusão de Curso – Bacharelado em Engenharia de Software, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2018.

Todos os alunos dos cursos de graduação da Universidade Tecnológica Federal do Paraná (UTFPR) devem realizar atividades complementares como disciplina obrigatória em seus currículos. Devido às dificuldades enfrentadas pelos alunos em gerenciar seu desempenho na disciplina de Atividades Complementares e pelo professor responsável por essa disciplina em avaliar, pontuar e em fazer um relatório com progresso e o desempenho de cada aluno durante o semestre, este trabalho se propõe a desenvolver uma plataforma web, onde os alunos de graduação da UTFPR possam armazenar seus documentos comprobatórios de participação em atividades complementares, preenchendo as informações inerentes a cada um desses documentos bem como quantos os pontos que suponham obter com cada documento. A partir dessas informações, o sistema determinará um relatório geral de quantos pontos o aluno supõem já ter atingido e com o provável progresso do aluno na disciplina. Finalizado o semestre, o aluno poderá enviar, por meio da plataforma, esses documentos ao professor para que este possa aprová-los ou não.Este projeto será desenvolvido pelo *framework* de processo de desenvolvimento Scrum Solo, usando o TDD (*Test Driven Development*) e o conceito de *web components*. O frontend será implementado pela biblioteca Vue usando HTML, CSS e JavaScript, e o backend, em Node com banco de dados MongoDB.

Palavras-chave: Atividade Complementares, *Web Components*, Scrum Solo, Aplicação Web

ABSTRACT

ROLIM, Rodrigo. Sistema web para gerenciar a pontuação dos alunos graduandos na disciplina de Atividades Complementares. 33 f. Trabalho de Conclusão de Curso – Bacharelado em Engenharia de Software, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2018.

All undergraduate students of the Universidade Tecnológica Federal do Paraná (UTFPR) should work complementary activities as a compulsory subject in their curricula. Due to the difficulties faced by students in managing their performance in the subject of Complementary Activities and by the teacher responsible for this discipline in assessing, scoring and reporting on progress and performance of each student during the semester, this paper proposes to develop a web platform, where undergraduate students of UTFPR can store their supporting documents for participation in complementary activities, filling in the information inherent to each of these documents as well as how many points they suppose to obtain with each document. From this information, the system will determine a general report of how many points the student supposes have already reached and with the probable progress of the student in the discipline. At the end of the semester, the student will be able to send these documents to the teacher through the platform so that they can approve them or not. This project will be developed by the Scrum Solo development process framework, using TDD (Test Driven Development) and the concept of web components. The frontend will be implemented by the Vue library using HTML, CSS and JavaScript, and the backend, in Node with MongoDB database.

Keywords: www, wwww, www

LISTA DE FIGURAS

LISTA DE ACRÔNIMOS

TDD	*Test Driven Development*
NoSql	*No Structured Query Language*
ASCII	*American Standard Code for Information Interchange*
SQL	Strutured Query Language
NPM	*Node Project Management*
UML	*Unified Modeling Language*
HTML	*Hypertext Markup Language*
CSS	Cascading Style Sheets

SUMÁRIO

1 INTRODUÇÃO

A Universidade Tecnológica Federal do Paraná (UTFPR), uma instituição pública federal, oferece mais de 121 cursos de graduação. Em todos estes cursos, a disciplina de Atividades Complementares é uma disciplina obrigatória e, portanto, todo aluno deve cumpri-la até o final do seu curso.

Atividades Complementares são atividades adicionais, paralelas às demais atividades acadêmicas, consideradas parte integrante do currículo e obrigatórias para a graduação do aluno. O objetivo das Atividades Complementares é instigar o aluno a participar de atividades que favoreçam o desenvolvimento de comportamentos sociais, humanos, culturais e profissionais conforme dito em (MOODLE, 2018).

A disciplina de Atividades Complementares é prevista e regulamentada pelo Regulamento das Atividades Complementares dos Cursos de Graduação da UTFPR, o (COEPP, 2006). O Regulamento determina que o aluno deve obter um mínimo de 70 pontos, ou 180 horas, em Atividades Complementares para conseguir aprovação em tal disciplina.

As Atividades Complementares são agrupadas em três grupos: ”Grupo 1 - atividades de complementação da formação social, humana e cultural; Grupo 2 - atividades de cunho comunitário e de interesse coletivo; e Grupo 3 - atividades de iniciação científica, tecnológica e de formação profissional”(PARANÁ-UTFPR,), onde, em cada grupo, aluno deve obter o mínimo de horas ou pontos.

Cabe ao aluno reunir e apresentar ao professor responsável pela disciplina de Atividades Complementares documentos que comprovem a sua participação em Atividades Complementares durante a sua graduação. Cada curso de graduação tem seu próprio professor que é responsável pela disciplina de Atividades Complementares.

Os documentos devem estar separados por grupo de atividade complementar, e cabe aos professores responsáveis da disciplina avalia se tal documento está em conformidade ou não com as normas do regulamento (COEPP, 2006).

1.1 OBJETIVOS

1.1.1 OBJETIVO GERAL

Desenvolver um sistema web, com base no Scrum Solo, no TDD (*Test Driven Developer*), no conceito de *web components* e no MVP*Minimum Viable Product* (*Minimum Viable Product*), que possibilite ao aluno um local online para guardar seus documentos comprobatórios de participação em Atividades Complementares e, ao fim do semestre, enviá-los ao professor responsável pela disciplina para que este possa ver e avaliar tais documentos. O sistema também fornecerá uma pontuação prévia (antes do professor avaliar) para o aluno na disciplina de Atividades Complementares.

1.1.2 OBJETIVOS ESPECÍFICOS

- Definir cronograma e planilha de custo para o projeto;
- Levantar requisitos de software: os funcionais e os não – funcionais de forma a atender as necessidades do cliente;
- Fazer uma análise desses requisitos;
- determinar a *Product Backlog*;
- Modelar a aplicação com base em uma arquitetura monolítica;
- Prototipar telas;
- Fazer um repositório no Github onde será hospedado e versionado o código-fonte;
- Fazer testes unitários automatizados;
- Implementar o código-fonte do software;
- Disponibilizar produto final em um servidor;
- *open-source* a fim de apenas demonstrar a aplicação funcional;
- Definir Ata de validação junto com o cliente.

1.2 ORGANIZAÇÃO TEXTUAL

O texto deste Trabalho está distribuído entre capítulos, seções e subseções, mais especificamente são cinco capítulos, vinte uma seções e cinco subseções. O primeiro capítulo faz uma introdução ao trabalho, versando sobre o conceito fundamental de atividades complementares. O segundo capítulo trata-se do referencial teórico, que está dividido em 17 seções. O terceiro capítulo é dedicado à descrição da metodologia utilizada no presente trabalho, esse capítulo tem uma seção (**Seção 3.1**) que, por sua vez, é onde está definido o cronograma deste trabalho. O quarto capítulo é o Desenvolvimento, onde se encontrar os requisitos do software levantados e o quinto capítulo estão escritas as considerações finais. No fim deste trabalho, no capítulo 5, tem-se as referências utilizadas. O quarto capítulo é o Desenvolvimento, onde se encontrar os requisitos do software levantados e o quinto capítulo estão escritas as considerações finais. No fim deste trabalho, no capítulo 5, tem-se as referências utilizadas.

2 REFERENCIAL TEÓRICO

Este capítulo fara uma breve explicação dos conceitos e ferramentas que serão utilizados neste trabalho para desenvolver o pretendido sistema *web*.

2.1 SCRUM SOLO

Scrum Solo é um *framework* de processo de desenvolvimento iterativo e incremental criado em 2012 por (PAGOTTO et al., 2016). Este *framework* é baseado no Scrum tradicional, porém adaptado para equipes de desenvolvimento com um único desenvolvedor. Basicamente o Scrum solo usa o Scrum tradicional, eliminando ou mudando algumas partes do processo. Vide a Figura 1, onde é possível ver que o Scrum Solo não tem as reuniões diárias s(o *daily scrum*) e as *sprints* duram no máximo uma semana (PAGOTTO et al., 2016).

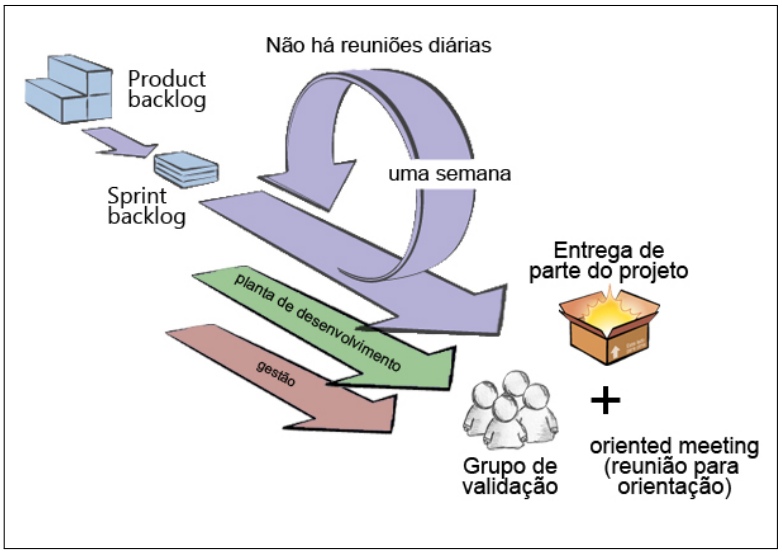

Figura 1: Scrum Solo

Fonte: (PAGOTTO et al., 2016)

A estrutura do Scrum Solo contém três particularidades, segundo (PAGOTTO et al., 2016): Atividades, Atores e Artefatos do processo. Cada Atividade do processo tem um con-

junto de artefatos de entradas e de saídas resultantes (artefatos do processo) e um conjunto de envolvidos (Atores do processo). Os envolvidos (cliente, desenvolvedor e orientador) são os mesmos em todas as atividades do processo.

2.1.1 ARTEFATOS DO PROCESSO

Os artefatos envolvidos no Scrum Solo são: Escopo, Protótipo do Software, *Product Backlog*, *Sprint Backlog*, Repositório do Processo, Produto, Ata, Planta de Desenvolvimento, Estrutura Analítica do Projeto (EAP), Cronograma e Planilha de Custo (PAGOTTO et al., 2016). A seguir, uma breve explicação sobre cada artefato.

O Escopo é um artefato que busca caracterizar o escopo do processo, no documento deste artefato consta o perfil do cliente (o interessado no software) e a lista com os itens que compõem a *Product Backlog* (PAGOTTO et al., 2016).

O Protótipo do Software é um artefato constituído pelo conjunto de protótipos de telas que fazem uma representação estática das *views* da interface de usuário, é importante que cada tela esteja apontada para o item da *Product Backlog* correspondente (PAGOTTO et al., 2016).

O *Product Backlog* é o conjunto de funcionalidades que devem ser implementadas no software, cada funcionalidade deve constar informações como o código da funcionalidade, a descrição, a data da inserção e a data de escolha da mesma para a *Sprint Backlog* (**??**).

O Repositório do processo é um serviço em nuvem de armazenamento de arquivos, o objetivo é guardar todos os artefatos gerados pelo processo nesse repositório (PAGOTTO et al., 2016).

O *Sprint Backlog* é o conjunto de itens selecionados da *Product Backlog* para serem implementados na *Sprint* (PAGOTTO et al., 2016).

O Produto é uma versão do software que possibilita ao cliente obter um retorno sobre o investimento feito na compra do software (PAGOTTO et al., 2016).

A Ata é o artefato cuja finalidade é fichar a validação da funcionalidade implementada. A Ata é uma lista com todas as funcionalidades implementadas no software. Para cada funcionalidade da lista, o cliente diz se está de acordo ou não com a implementação (PAGOTTO et al., 2016).

A Planta de Desenvolvimento é a reunião de todos os diagramas em UML usados para definição e modelagem de todas as funcionalidades do software. Segundo (PAGOTTO et al., 2016), esse artefato se caracteriza pelos seguintes diagramas: caso de uso, de sequência, de

classes e entidade e relacionamento.

A Estrutura Analítica do Projeto (EAP) é um organograma que busca representar o escopo do projeto, o qual é dividido em componentes menores e mais gerenciáveis. Isto é, as atividades que definem o escopo do projeto são quebradas em partes substanciais (PAGOTTO et al., 2016).

O Cronograma serve para organizar os pacotes de trabalho em uma sequência com relação a um espaço de tempo determinado (PAGOTTO et al., 2016).

Por último, a Planilha de Custo, que servirá para mapear o custo efetivo do projeto durante a execução do mesmo. Ao final do projeto, tem-se uma visão consistente do que foi orçado com o que foi realizado (custo e tempo real)(PAGOTTO et al., 2016).

2.1.2 ATORES DO PROCESSO

Os Atores do Processo Scrum Solo são três: Cliente, que é o interessado no produto de *software*; desenvolvedor, responsável por aplicar o processo e construir o *software*; e o orientador, que tem uma visão vasta e profunda do processo, do escopo de produto e das tecnologias usadas para desenvolver o produto (PAGOTTO et al., 2016).

2.1.3 ATIVIDADES DO PROCESSO

São exatamente quatro atividades que compõem o processo do Scrum Solo, os quais são: *Requirement*, *Sprint*, *Deployment* e *Management*.

O *Requirement* é a primeira atividade, a qual consiste em definir o escopo do produto, caracterizar o cliente do produto e determinar a *Product Backlog* (grupo de funcionalidades ou requisitos que o sistema tem que ter). É nessa fase que um conjunto de informações sobre o pontencial sistema são coletadas do cliente e do orientador os quais são os artefatos de entrada dessa atividade. Por sua vez, Os artefatos gerados são: Escopo, Product Backlog, Protótipo do *Software* (PAGOTTO et al., 2016).

A *Sprint* é a segunda atividade do processo, cujo objetivo é estabelecer a *Sprint Backlog*, isto é, uma lista de requisitos escolhidos da *Product Backlog* para serem implementados na *Sprint* em, no máximo, uma semana. Os artefatos gerados nessa etapa são: *Sprint Backlog* (subconjunto extraído do *Product Backlog* para rodar no *life clicle* da *sprint*), Produto, Ata e Planta de desenvolvimento (PAGOTTO et al., 2016).

Deployment é a atividade na qual é disponibilizado o produto final ao cliente para uso.

O artefato de entrada nessa atividade é a planta de desenvolvimento e os artefatos gerados são: Produto e Ata (PAGOTTO et al., 2016).

Management, quarta e última atividade do processo Scrum Solo, que tem como objetivos Planejar, monitorar e controlar o desenvolvimento do produto. Os artefatos de entrada é a *Product Backlog*, e os de saída são: Estrutura Analítica do Projeto (EAP), Cronograma, Planilha de Custo, Planilha de Controle. Todos os artefatos gerados em todas essas atividades devem ser armazenados no repositório do processo (PAGOTTO et al., 2016).

2.2 MVP

De acordo com (ENDEAVOR, 2018), o MVP (*Minimum Viable Product*), que significa Mínimo Produto Viável, é uma técnica de programação que visa entregar o *software* somente quando o mesmo atender minimamente todos os requisitos necessários. Em outras palavras, é entregue uma versão mais ”enxuta”do *software*, mas, a qual, atende a todos os anseios do cliente (ENDEAVOR, 2018). A **Figura 2** busca ilustrar o sentindo do MVP.

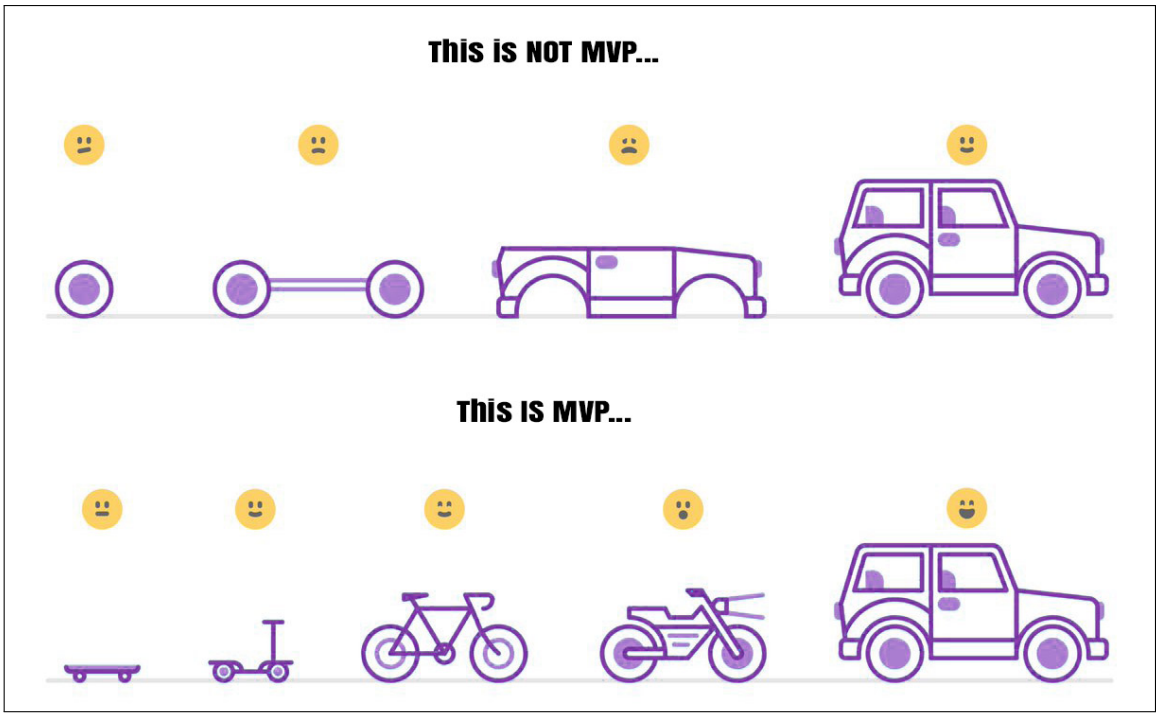

Figura 2: ***Ilustração do MVP***

Fonte: (ENDEAVOR, 2018)

2.3 GITHUB E GIT

Git é um sistema de controle de versão *open-source* de arquivos, designado para gerenciar quaisquer projetos. Por meio dele é possível desenvolver projetos nos quais diversas pessoas podem contribuir sem o risco de ocorrer sobrescrições entre trabalhos de programadores diferentes (GIT, 2018).

Github é uma serviço ou plataforma *web* gratuita de hospedagem e para controle de versão de projetos (GITHUB, 2018). No Github, o projeto é gerenciável e o código pode ser hospedado e revisto por meio de comandos do Git (**??**).

2.4 TDD

O TDD (Test Driven Developement) é uma técnica de programação orientada a teste automatizados. Neste sentido, os testes automatizados são desenvolvidos ainda antes de implementar o *software*. Para cada componente da aplicação que é construída, um conjunto de testes são escritos antes do desenvolvimento para garantir que a aplicação funcione conforme o almejado (LOPES, 2012).

2.5 TRAVIS CI

Travis CI é uma plataforma online de integração contínua *open-source* que suporta o processo de desenvolvimento através da criação e teste de mudanças de código. Os teste acontecem por meio de condições pre-definidas pelo próprio programador (quando ocorre uma mudança de código, por exemplo) no *script* `.travis.yml` do Travis-CI, esse *script* fica hospedado no GitHub, acionando os testes sempre que as condições descritas nele forem satisfeitas (TRAVIS-CI, 2018).

2.6 PLANTUML

PlantUML é uma ferramenta *open-source* para criar diagramas em UML a partir de uma linguagem de texto simples (PLANTUML, 2009), o que o torna ideal para hospedar os diagramas em um repositório e, dessa forma, versioná - los através do Git.

2.7 *VUE*, *VUE-RESOURCE* E *VUE-ROUTER*

Vue (pronuncia-se *view*) é biblioteca em *JavaScript* para desenvolver componentes reativos para *web*. Basicamente, componentes reativos são frações elementares de código constituídos com linguagem de marcação (*html*), de estilo (*css*) e de comportamento (*JavaScript*) próprios. Um conjunto dessas frações de código compõem uma página, de modo que torna fácil e elegante o reaproveitamento de código (INCAU, 2017, 1).

Um componente é dito reativo quando sua estrutura de marcação (*html*) reagi e se adaptar adequadamente ás mudanças de comportamento desse componente (por meio do *JavaScript*). Em outras palavras o código em *html* é alterado e moldado conforme as ordens oriundas do correspondente código em *JavaScript* (INCAU, 2017, 2).

Segundo (REIS, 2016) o *vue* é um tecnologia que tem ascendido nos últimos anos. No ano de 2016, o número de downloads do *Vue* ultrapassou 1 milhão pelo NPM. Grandes empresas usam o *Vue* como tecnologia de desenvolvimento do *frontend*, como, por exemplo, as empresas chinesas: Alibaba, Baidu e Tecent.

O *Vue-Resource* é uma biblioteca feita em *JavaScript* usada para fazer requisições externas por meio do protocolo HTTP. O Vue-Resource executa suas requisições usado a *Promise* API (INCAU, 2017, 105). O *Vue* utliza, por padrão, essa biblioteca para fazer requisições assíncronas.

2.8 VUEX

2.9 KARMA, MOCHA, CHAI E SINON

Nesta seção, serão dados brevemente os conceitos de cada uma das ferramentas que, juntas, serão usadas para realizar os testes unitários automatizados, os quais são: Karma, Chai, Mocha e Sinon. Todas são bibliotecas ou *frameworks* escritas em *javascript* que serão integradas para definição dos testes.

Karma é open-source. Ele não é *framework* de testes, nem serve para fazer verificações (ou asserções). Ele apenas cria um servidor HTTP e gera o *html* a ser testado do *framework* de teste (INCAU, 2017, 59).

O objetivo do Karma é garantir ao desenvolvedor um ambiente de teste muito mais produtivo, de modo que o desenvolvedor não precise se preocupar com um monte configurações necessárias para realizar os testes (INCAU, 2017).

Karma fornece um ambiente onde os desenvolvedores podem escrever seu código e obter uma resposta.

Mocha é um *framework*, que roda em Node, de testes compreensíveis e maleáveis, repleto de funcionalidades (INCAU, 2017, 59). Esse *framework* simplesmente serve para criar os testes.

Chai é uma biblioteca de asserções *open-source* que ajuda a escrever testes de modo mais fácil e deixá-los mais legíveis (INCAU, 2017, 60).

Sinon É uma biblioteca para simular classes que ainda não foram implementadas, mas que são dependências de outras classes que precisam ser testadas. Muitas vezes, tem-se que testa uma classe que depende de outras classes que ainda não foram implementadas, o Sinon serve para fazer o *mock* dessas classes (INCAU, 2017, 60).

Desta forma, o Karma será útil para fornecer um ambiente onde os testes podem ser executados; o Mocha para criar os testes, sendo estes escritos em forma de asserções por meio da biblioteca Chai e aquelas classes necessárias, porém ainda não implementadas, podem ser simuladas com a ajuda da biblioteca Sinon (INCAU, 2017).

2.10 NODE E EXPRESS

Node é uma plataforma *runtime* escrito em *JavaScript*, designado para construir aplicações web escaláveis. E *Express* é um framework Node minimalista e flexível para construção de aplicações web, conforme dito em (EXPRESS, 2017).

2.11 NOSQL

Segundo (FOWLER, 2013) o termo "*NoSql*"surgiu em 1990 o que diz respeito a bancos que não utilizam o modelo relacional e não possuem esquemas como no bando de dados relacional, por isso é conhecido como um banco de dados não-relacional. O banco de dados *NoSql* armazena suas relações sob o formato de aquivos ASCII, onde cada registro é representado por uma linha com os valores separados por tabulações.

O nome *NoSql* vem do fato de que esse tipo de banco não usa comandos SQL para realizar consultas, em vez disso, ele usa *scripts shells* que podem ser encadeados (*pipelines*) no Unix conforme descrito em (FOWLER, 2013).

2.12 MONGODB

O MongoDB é um banco de dados não-relacional orientado a documentos, pois ele armazena e recupera documentos, os quais podem ser JSON, XML, BSON, entre outros. Tais documentos são estruturas de dados em árvores hierárquicas e autodescritivas, constituídas de mapas, coleções e valores escalares. Bancos de dados orientados a documento funcionam com base no conceito de chave-valor, onde o valor pode ser examinado. Veja um exemplo de estrutura de documento em MongoDB na Figura 3, que é simplesmente um conjunto de pares chave-valores separados por vírgulas.

```
{
   _id: ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100,
   comments: [
      {
         user:'user1',
         message: 'My first comment',
         dateCreated: new Date(2011,1,20,2,15),
         like: 0
      },
      {
         user:'user2',
         message: 'My second comments',
         dateCreated: new Date(2011,1,25,7,45),
         like: 5
      }
   ]
}
```

Figura 3: Estrutura de documento em MongoDB

Fonte: (TUTORIALSPOINT, 2018)

2.13 NPM

Conforme dito em (NPM, 2018) o NPM é um gerenciador de dependências de projetos em *JavaScript* que facilita a reutilização e compartilhamento de códigos *JavaScript*. E o NPM é distribuído com o *Node*.

2.14 WEBPACK

O *webpack* é um empacotador de módulos estáticos para aplicativos *JavaScript* modernos. Ele serve para agrupar os recursos usados em uma aplicação em pacotes correspondentes. Ao processar aplicativos, o *webpack* cria internamente um gráfico de dependências que mapeia

todos os módulos que seu projeto precisa e gera um ou mais pacotes configuráveis (WEBPACK, 2018).

Basicamente, a configuração do *webpack* consiste de quatro conceitos: *entry*, *output*, *loaders*, *plugins*.

O *entry* ou *entry point* indica qual módulo o *webpack* deve começar a construir seu gráfico de dependência interno. O *webpack* irá descobrir quais outros módulos e bibliotecas que o *entry point* depende.

O caminho e valor do arquivo que configura o *entry point* por padrão é `./src/index.js`. Veja na Figura 4 a configuração exemplo do *entry point*.

Figura 4: Configuração do *entry point*
Fonte: (WEBPACK, 2018)

A propriedade *output* informa ao *webpack* onde emitir os *bundles* que ele cria e como nomear esses arquivos. Veja um exemplo de cofiguração do *output* na Figura **??**.

Figura 5: Configuração do *output*
Fonte: (WEBPACK, 2018)

O *webpack* apenas compreende arquivos *JavaScript*. A propriedade *loaders* permiti ao webpack a capacidade de processa outros tipos de arquivos e convertê-los em módulos válidos que possam ser consumidos pela aplicação e adicionados no gráfico de dependência dessa aplicação. A Figura 6 demonstra um exemplo de configuração dessa propriedade em específico.

webpack.config.js

```
const path = require('path');

module.exports = {
  output: {
    filename: 'my-first-webpack.bundle.js'
  },
  module: {
    rules: [
      { test: /\.txt$/, use: 'raw-loader' }
    ]
  }
};
```

Figura 6: Configuração do *loaders*

Fonte: (WEBPACK, 2018)

Plugins podem ser usada para realizar uma grande número de tarefas, como otimização de *bundle*, gerenciamento de *assets* e injeção de variáveis de ambiente. A Figura 7 dá um exemplo de configuração da propriedade *plugins* do *webpack*.

webpack.config.js

```
const HtmlWebpackPlugin = require('html-webpack-plugin'); //installed via npm
const webpack = require('webpack'); //to access built-in plugins

module.exports = {
  module: {
    rules: [
      { test: /\.txt$/, use: 'raw-loader' }
    ]
  },
  plugins: [
    new HtmlWebpackPlugin({template: './src/index.html'})
  ]
};
```

Figura 7: Configuração do *plugins*

Fonte: (WEBPACK, 2018)

2.15 HEROKU

Heroku é uma plataforma de serviços em nuvem ou PaaS*Platform as a Service* (Platform as a Service) que possibilita desenvolvedores a construir, executar, e operar totalmente as aplicações na nuvem. Essa plataforma é baseada em sistema de *containers* gerenciados, com serviços de dados integrados e um poderoso ecossistema para implantar e executar aplicações modernas (HEROKU, 2018).

O Heroku segue uma abordagem de integração e entrega contínuas de *software* (HEROKU, 2018), e é uma excelente plataforma para demonstrar uma aplicação *web* em funcionamento.

3 METODOLOGIA

Para desenvolver o pretendido sistema, serão realizadas os passos expressos como objetivos específicos na seção de objetivo do trabalho, usando o processo de desenvolvimento Scrum Solo.

A primeira etapa é o levantamento de requisitos, onde acontecerão conversas diretas entre o desenvolvedor e o cliente (Atores do processo) interessado no produto de *software* a fim de se extrair dessa comunicação informações que representem os requisitos ou as funcionalidades do *software*. A definição do perfil do cliente bem como dos requisitos defini o Escopo do projeto e do produto.

Somente após definir o escopo do produto, será determinado um cronograma para as atividades que comporão o processo de desenvolvimento do produto.

Os requisitos serão agrupados e categorizados em conjuntos conforme um critério: se os requisitos estão correlacionados de modo que, se forem implementados, resulte em um componente da aplicação *web*, então devem ser reunidos em um conjunto específico. Esses conjuntos obtidos devem ser listados e registrados como sendo a *Product Backlog*. Cada item da *Product Backlog* é um conjunto daqueles.

Para todos os itens da *Product Backlog*, será construído um conjunto de quatro diagramas de UML (*Unified Modeling Language*), feitos com a ferramenta PlantUML, que definem de forma pictográfica a codificação das informações subjacentes de cada um desses itens da *Product Backlog*. Esses diagramas são: diagrama de caso de uso, de sequência, de classes e de entidade e relacionamento.

Esse conjunto gerado juntamente com os protótipos de tela, os quais serão feitos com a mesma biblioteca que vai ser usada para implementar o *frontend* (o *Vue*), é a planta de desenvolvimento. Ambos, *Product Backlog* e a planta de desenvolvimento, serão armazenados em um repositório em nuvem (o GitHub) e versionados, usando comandos do Git.

Em seguida, a partir da *Product Backlog*, é selecionado um grupo de itens que com-

porão a *Sprint Backlog* de tal modo, que a execução de um item durante a *Sprint* resulte em novo componente da aplicação *web*, isto é um componente com HTML, CSS e *JavaScript* próprios.

A **Figura 3** dá uma visão representativas dos itens resultando em componentes. Cada item da *Sprint*, terá um anexo com uma parcela dos diagramas UML que o definem e com protótipos de telas relativos a esse item. Para cada item da *Sprint Backlog* é escrito, antes mesmo de implementar o código da funcionalidade que o item representa, casos de testes automatizados para testar o código que implementará o item.

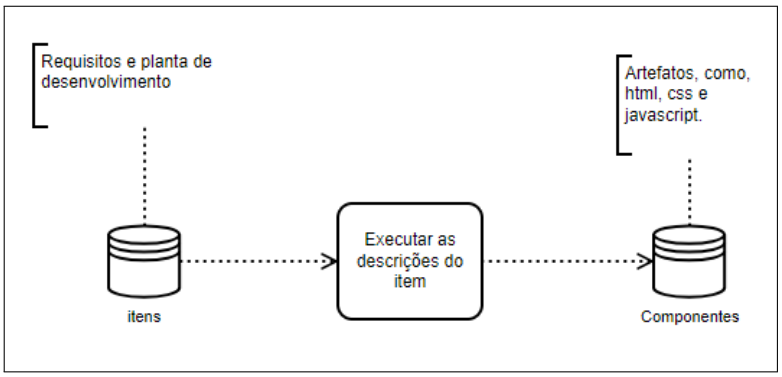

Figura 8: A execução do item resulta em um componente
Fonte: Autoria própria

Os itens da *Product Backlog* serão dispostos em tarefas no *project boards* (quadro de projeto) disponível no próprio GitHub na sua aba de *Project*, que é simplesmente a implementação de um kanban. Neste quadro de projeto, definir-se-á seis colunas:*In box*, *To do*, *In progress*,*Review* e *Done*. No próximo paragrafo, é esclarecido a finalidade de cada uma dessas colunas.

A coluna *In box* contém todos os itens da *Product Backlog*. A coluna *To do* contém os itens selecionados para serem implementadas em uma dada *Sprint*, esse grupo de itens é chamado de *Sprint Backlog*. A Coluna *In progress* é o estado em que a tarefa está em progresso, ou seja, em desenvolvimento. A coluna *Review* contém as tarefas que estão em fase de teste, os testes são automaticamente executados através dos *scripts* configurados no arquivo `.travis.yml` do *Travis-CI*, se o item implementado passar pelos testes, a tarefa relativa ao item deve ser arrastada para *Done*. Mas se a implementação de um item não passar pelos testes, a tarefa estaciona em *Review*, e tal implementação reprovada deve ser refatorado até passar em todos os testes. A coluna *Done* contém as tarefas já implementadas e que foram aprovadas pelos testes. Veja a Figura 9.

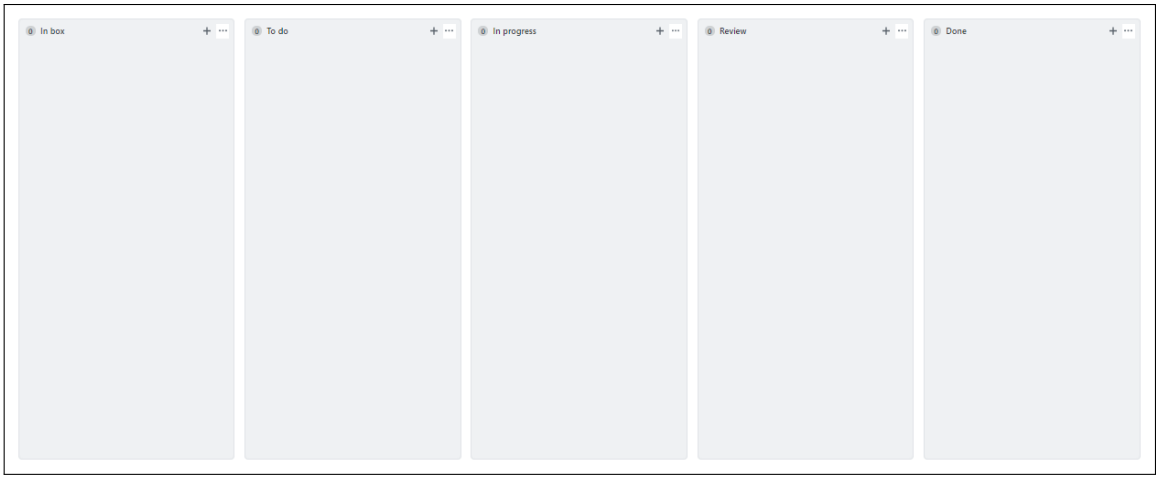

Figura 9: *Project Board* do GitHub que será usado neste trabalho

Fonte: Autoria própria

A execução de um item resulta em um componente *web*. Cada componente é constituído de elementos do *frontend* e do *backend*. Todos os componentes obtidos ficarão armazenados no repositório do GitHub.

Para obter o produto final, os componentes passam por um processo de integração, onde os componentes serão unidos uns com os outros. Se um componente se comunica com outro, ou seja, de alguma forma trocam informações entre si, estes dois devem ser integrados implementando essa comunicação entre eles.

O arrastar das tarefas entre as colunas será realizada pelo próprio desenvolvedor. Toda vez que uma *issue* é aberta, uma nova tarefa no *project board* do trabalho no GitHub é adicionada, nessa tarefa serão anexadas arquivos que contém a Planta de Desenvolvimento relativa ao item da tarefa.

Esse conjunto de procedimentos explicados anteriormente, são repetidos até que, de um lado, esvazie-se a *Product Backlog*, ou seja, não haja mais itens para serem implementados e, do outro, tenha - se um produto minimamente viável (MVP) pronto para entregar ao cliente e o conjunto de componentes obtidos são armazenados no repositório em banco de componentes, veja na **Figura 2**.

Agora, tendo em mãos todos os componentes minimamente implementados, os componentes passarão por sucessivos ciclos incrementais de melhorias, sempre obtendo uma nova versão melhorada do *software*. Ao fim de cada ciclo, o conjunto de componentes é atualizado com os mesmos componentes, porém refinados pelas melhorias realizadas. As fases desse processo são as mesmas do processo de fabricação dos componentes. Veja a Figura 10 para ter uma

visão geral desses procedimentos explicitados.

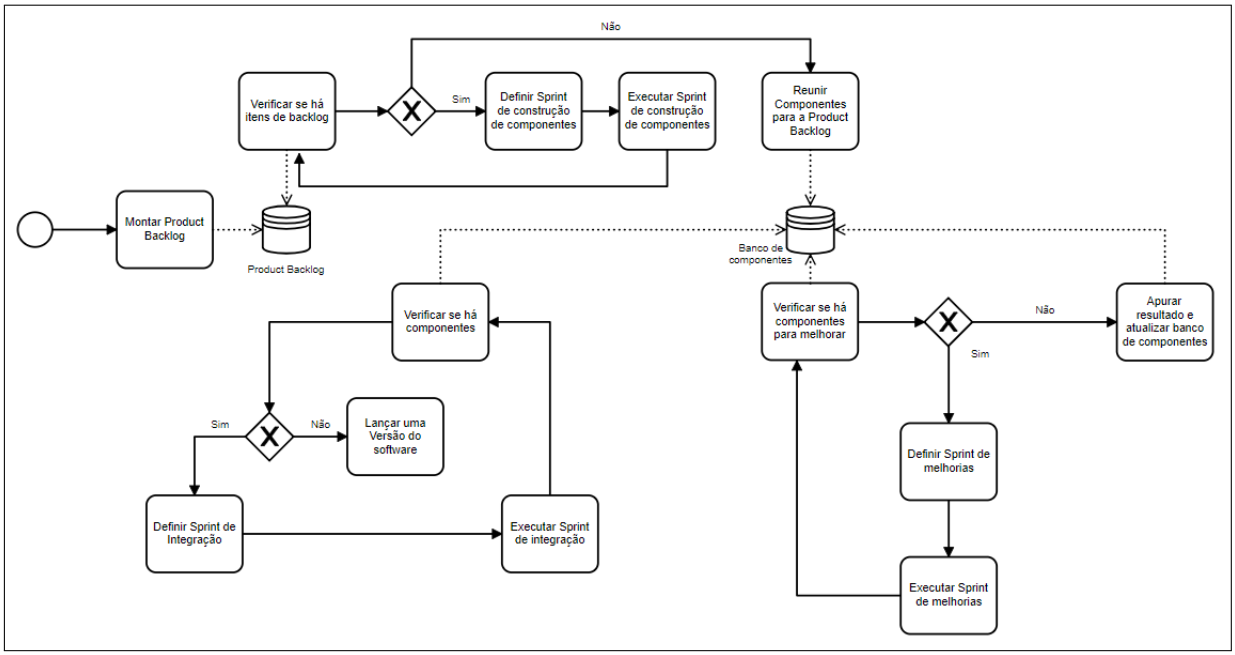

Figura 10: Fluxograma de como acontecerá o desenvolvimento

Fonte: Autoria própria

O *frontend* da aplicação será feito com a biblioteca *Vue*, onde serão construídos os componentes que constituem cada uma das *views* da aplicação bem como seu respectivo *backend*. O projeto *Vue* será criado e configurado com o *webpack* por meio do comando: `vue init webpack <nome do projeto>`.

As ferramentas que serão usadas em conjunto para dar vida aos testes unitários automatizados são: *Karma*, que é um espécie de *runner* dos testes, o comando para instalá-lo é `npm install karma@version`; *Chai*, que serve para fazer as asserções de teste (instalado pelo comando: `npm i chai`); *Mocha*, para criar os testes, o comando para instalá-lo é `npm i mocha`; e o *Sinon* para simulação de dependências ainda não implementadas para efetivar os testes, o comando para instalá-lo no projeto é `npm i sinon`. Os teste unitários automatizados serão implementados somente (no *frontend*), para o *backend* não serão feitos testes unitários automatizados.

O *backend* será implementado através da plataforma Node juntamente com o *framework Express* que será usado para tratamento e exibição dos dados. Essas duas tecnologias receberão as requisições oriundos das interações dos usuários com as *views* e, de volta, enviará as respostas adequadas. Essas são as requisições/respostas HTTP que, no *Vue*, são manipuladas

pela biblioteca *Vue-resource*.

Como banco de dados da aplicação, este trabalho usará o MongoDB, que é um banco não-relacional orientado a documentos. A modelagem desse banco é pouco diferente dos bancos relacionais, pois são feitos com representações de dados estruturados, ou melhor, de agregados. Deste modo, o modelo deste banco será a modelagem, porém desnormalizada, do banco que segue uma orientação relacional.

Finalizado o processo, será feito uma avaliação de todo o processo, verificando o custo real dos procedimentos que é, então, comparado ao custo estipulado, e a validação da lista de funcionalidades implementadas junto ao cliente. Essa lista é a Ata, um dos artefatos especificados no Scrum Solo.

3.1 CRONOGRAMA

No cronograma, além de outras atividades, a *Sprint 1* é onde serão construídos os componentes com base na descrição de cada um dos itens da *Product Backlog*; *Sprint 2* tem como entrada os componentes gerados na *Sprint 1*, os componentes serão integrados; e na *Sprint 3*, serão feitas melhorias nesses componentes.

Tabela 1: Cronograma de Atividades.

Atividades	**período**
1. Criar repositório no github	16 de julho
2. Levantamento de requisitos	17 a 20 de julho
3. Definir Project Backlog	22 a 25 de julho
4. Desenhar diagramas UML	22 a 25 de julho
5. Fazer protótipos de tela	26 a 27 de julho
6. Planejar sprint 1	10 de agosto
7. Executar sprint 1	11 a 17 de agosto
8. Reunir banco de componentes	18 de agosto
9. Planejar sprint 2	19 de agosto
10. Executar sprint 2	20 a 26 de agosto
11. Planejar sprint 3	28 de agosto
12. Executar sprint 3	29 de agosto a 4 de setembro
13. Fichar Ata de validação	8 a 12 de setembro
14. Realizar deploy da aplicação	18 de setembro

4 DESENVOLVIMENTO

Neste capítulo serão demonstrados os requisitos funcionais e não-funcionais inicialmente levantados. Os requisitos funcionais estão divididos em duas categorias conforme o tipo de usuário do sistema: as que são relativas à sessão do professor responsável e os que tem haver com a sessão do aluno. Os requisitos não-funcionais vêm em seguida aos requisitos funcionais.

4.1 REQUISITOS FUNCIONAIS

Tabela 2: Requisitos Funcionais da sessão de professor

Código	Descrição	Prioridade	Relacionados
RF01	O sistema deve ter uma caixa de entrada onde professor responsável receberá os documentos dos alunos.	Essencial	RF02
RF02	O professor acessará essa caixa de entrada e poderá fazer a análise desses documentos, podendo aprová ou reprová-los conforme as normas.	Essencial	
RF03	Na caixa de entrada do professor, os documentos serão agrupados por aluno (o remetente)	Desejável	RF04
RF04	Um conjunto de documentos de um aluno deve conter os documentos organizados por grupo de atividade complementar	Desejável	RF03
RF05	O sistema deve permitir que o professor se cadastre no próprio sistema	Essencial	
RF06	O sistema deve permitir que o professor faça login no próprio sistema	Essencial	
RF07	O sistema deve permitir que o professor possa alterar seus dados cadastrais.	Importante	RF05

Tabela 3: Requisitos Funcionais da sessão de alunos

Código	Descrição	Prioridade	Relacionados
RF08	O sistema deve permitir que o aluno possa se cadastrar no sistema	Essencial	
RF09	O sistema deve permitir que o aluno possa fazer login no sistema	Essencial	
RF10	O sistema deve permitir que o aluno altere seus dados cadastrais	Desejável	
RF11	Um documento, após ser enviado pelo aluno, terá um dos três status, que serão exibidas para o aluno: aguardando análise, devolvido ou aprovado	Desejável	RF12
RF12	O status do documento, exibido para o aluno, é definido conforme o professor aprove ('Aprovado'), reprove ('Devolvido') ou se o documento ainda não foi avaliado ('Aguardando análise').	Importante	
RF13	Na sessão do aluno, os documentos devem ser separados conforme o grupo de atividade complementar a que pertencem.	Desejável	
RF14	Em cada grupo, o documento contém sua informações descritivas	Desejável	RF13

4.2 REQUISITOS NÃO-FUNCIONAIS

Tabela 4: Requisitos Não Funcionais

Código	Descrição	Categoria
RNF01	O sistema deve ser disponibilizado na *web* e deverá possuir um design responsivo	Usabilidade
RNF02	O sistema deve funcionar perfeitamente nos seguintes navegadores: Mozilla, FireFox e Chrome	Compatibilidade
RNF03	O sistema deve realizar transações de dados com o banco de dados não relacional MongoDB	Interoperabilidade
RNF04	O sistema fará a autenticação do cadastro dos usuários (alunos e professor) por meio dos emails institucionais dos mesmos, assim é provado que o usuário que esta tentando fazer cadastro faz parte do quadro acadêmico da instituição federal (UTFPR).	Segurança

Tabela 5: Requisitos Não-Funcionais

Código	Descrição	Categoria
RNF05	Todas as entradas de usuários passarão por validações afim de evitar prejuízos a segurança do sistema	segurança
RNF06	O sistema fará a autenticação do login dos usuários por meio da senha e nome de usuário (RA ou SIAPE) dos mesmos. Assim, cada usuário terá uma sessão específica.	segurança
RNF07	O desenvolvimento do sistema se dividirá em três partes: o frontend, que será feito com a biblioteca Vue; o backend, que será feito com Node; e o banco de dados, por meio do MongoDB.	Padrão
RNF08	No módulo de envio de documentos pelo aluno, dados complexos como arquivos de texto e imagens serão armazenados em pastas, enquanto os nomes dos arquivos devem ser guardados no banco de dados.	Desempenho
RNF09	Os teste unitários devem ser automatizados	Testabilidade
RNF10	Na parte de upload de documento feito pelo aluno no momento de envio de documento o sistema deve suportar arquivos nos seguintes formatos: PDF e imagens PNG	Compatibilidade

5 CONCLUSÃO

A intenção final é disponibilizar um produto que seja fácil de usar tanto para o aluno quanto para o professor, facilitando o trabalho de ambos no que tange ao gerenciamento das Atividades Complementares. Os atributos de qualidade de software que o presente trabalho tentará respeitar são o desempenho e a manutenibilidade do software que se propõem a fazer. Além disso, este trabalho também adotará o Scrum Solo como processo de desenvolvimento.

Este software deverá ser um ponto de referência para os alunos quanto a acompanhar o seu desempenho em Atividades Complementares a cada semestre e para os professores responsáveis pela disciplina de Atividades Complementares no seu trabalho de administrar o progresso de todos os alunos nessa disciplina.

REFERÊNCIAS

CONSELHO DE ENSINO, PESQUISA E PÓS-GRADUACÃO. **Resolução nº 61/06 - CO-EPP**: Regulamento das atividades complementares dos cursos de graduação da utfpr. Paraná, june. 2006. 11 p.

ENDEAVOR. **O Guia Prático para o seu MVP – Minimum Viable Product**. 2018. Disponível em: <https://endeavor.org.br/mvp/>. Acesso em: 21 de maio de 2018.

EXPRESS. **Express**. 2017. Disponível em: <https://expressjs.com/>. Acesso em: 16 de maio de 2018.

FOWLER, P. J. S. M. **NoSQL Essencial Um Guia Conciso para o Mundo Emergente da Persistência Poliglota**. 1. ed. [S.l.]: Novatec, 2013.

GIT. **What is Git**. 2018. Disponível em: <https://www.atlassian.com/git/tutorials/what-is-git>. Acesso em: 17 de junho de 2018.

GITHUB. **GitHub**. 2018. Disponível em: <https://github.com>. Acesso em: 14 de maio de 2018.

HEROKU. **Heroku**. 2018. Disponível em: <https://www.heroku.com/>. Acesso em: 21 de maio de 2018.

INCAU, C. **Vue.js: construa aplicações incríveis**. 1. ed. São Paulo: Casa do Código, 2017.

LOPES, C. **Princípios de Test Driven Development (TDD)**. 2012. Disponível em: <https://imasters.com.br/artigo/24242/desenvolvimento/principios-de-test-driven-development-tdd/>. Acesso em: 2 de maio de 2018.

MOODLE. **DACOM-CP - Atividades Complementares**. 2018. Disponível em: <http://moodle.utfpr.edu.br>. Acesso em: 10 de abril de 2018.

NPM. **Get npm**. 2018. Disponível em: <https://www.npmjs.com/get-npm>. Acesso em: 23 de maio de 2018.

PAGOTTO, T. et al. Scrum solo: software process for individual development. In: IEEE. **Information Systems and Technologies (CISTI), 2016 11th Iberian Conference on**. [S.l.], 2016. p. 1–6.

PARANÁ-UTFPR, U. T. F. do. **Atividades Complementares**. Disponível em: <http://www.utfpr.edu.br>.

PLANTUML. **PlantUML**. 2009. Disponível em: <http://plantuml.com/>. Acesso em: 10 de maio de 2018.

REIS, V. **Como é a popularidade do Vue.JS no mercado?** 2016. Disponível em: <http://vuejs-brasil.com.br/como-e-a-popularidade-do-vue-js-no-mercado/>. Acesso em: 20 de junho de 2018.

TRAVIS-CI. **Travis-CI**. 2018. Disponível em: <https://travis-ci.org/>. Acesso em: 9 de maio de 2018.

TUTORIALSPOINT. **MongoDB - Overview**. 2018. Disponível em: <https://www.tutorialspoint.com/mongodb/mongodb$_{o}verview.htm¿.Acessoem$: 20$de\,junhode$2018.

WEBPACK. **Concepts**. 2018. Disponível em: <https://webpack.js.org/concepts/>. Acesso em: 23 de maio de 2018.