CENTRO UNIVERSITÁRIO DE ANÁPOLIS – UniEVANGÉLICA
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

DAVI GONÇALVES DE SOUZA
MARCOS MORAES PEREIRA

SISTEMA PARA PUBLICAÇÃO E COMPARTILHAMENTO DE TRABALHOS ACADÊMICOS – SEMEAR

ANÁPOLIS
2019

DAVI GONÇALVES DE SOUZA
MARCOS MORAES PEREIRA

SISTEMA PARA PUBLICAÇÃO E COMPARTILHAMENTO DE TRABALHOS ACADÊMICOS – SEMEAR

Trabalho de Conclusão de Curso II apresentado como requisito parcial para a conclusão da disciplina de Trabalho de Conclusão de Curso II do curso de Bacharelado em Engenharia de Computação do Centro Universitário de Anápolis – UniEVANGÉLICA.

Orientador(a): Prof. Me. Millys Fabrielle Araujo Carvalhaes.

Anápolis
2019

DAVI GONÇALVES DE SOUZA
MARCOS MORAES PEREIRA

SISTEMA PARA PUBLICAÇÃO E COMPARTILHAMENTO DE TRABALHOS ACADÊMICOS - SEMEAR

Trabalho de Conclusão de Curso II apresentado como requisito parcial para a obtenção de grau do curso de Bacharelado em Engenharia de Computação do Centro Universitário de Anápolis – UniEVANGÉLICA.

Aprovado(a) pela banca examinadora em 29 de novembro de 2019, composta por:

Millys Fabrielle Araújo Carvalhaes
Presidente da Banca

Luciana Nishi
Prof(a). Convidado(a)

Walquiria F Marins
Walquiria Fernandes Marins
Prof(a). Convidado(a)

RESUMO

No meio acadêmico são utilizados diversos métodos no processo de aprendizagem, dentre eles, elaboração de trabalhos; resumos e artigos. Este trabalho propõe realizar o desenvolvimento de uma ferramenta chamada SEMEAR, que possa auxiliar os alunos na publicação destes trabalhos acadêmicos, utilizando técnicas, ferramentas e tecnologias consistentes, que possibilitam a construção de um produto de *software* com qualidade. Neste trabalho será utilizado uma metodologia de desenvolvimento ágil pertencente a categoria de processos iterativos incrementais, a qual permite entregar valor ao cliente de forma rápida, a fim de minimizar erros sendo adaptativa a mudanças. O resultado deste projeto visa demonstrar as conformidades na execução das etapas de desenvolvimento, baseando-se no referencial teórico levantado durante a sua construção. Deste modo é proposto a execução e planejamento de métricas, metodologias e desenvolvimento prático a fim de obter um produto palpável no fim de sua execução.

Palavras-chave: Processo de aprendizagem, elaboração de trabalhos, publicação de trabalhos.

ABSTRACT

In the academic environment, several methods are used in the learning process, among them, elaboration of works; abstracts and articles. This work proposes the development of a tool called SEMEAR, which can assist students in the publication of these academic works, using consistent techniques, tools and technologies that enable the construction of a quality software product. In this work we will use an agile development methodology belonging to the category of incremental iterative processes, which allows to deliver value to the customer quickly, in order to minimize errors and be adaptive to changes. The result of this project aims to demonstrate the conformity in the execution of the development stages, based on the theoretical framework raised during its construction. Thus it is proposed the execution and planning of metrics, methodologies and practical development in order to obtain a tangible product at the end of its execution.

Keywords: *Learning process, elaboration of works, publication of works.*

LISTA DE ILUSTRAÇÕES

Figura 1 - Ilustração de um quadro Kanban 14
Figura 2 - Exemplo de uma História de Usuário 17
Figura 3 - Componente em React 20
Figura 4 - Componente em React sem props. 21
Figura 5 - Componente em React com props. 21
Figura 6 - Exemplo de componente *presentational* 22
Figura 7 - Exemplo do estado do componente recebendo 'World' 22
Figura 8 - Banco de dados Relacional versus Não Relacional 24
Figura 9 - Conexão com banco do Firebase 25
Figura 10 - Método que retornar a quantidade de estrelas de uma postagem 25
Figura 11 - Diagrama de Casos de Uso 27
Figura 12 – Estrutura do Banco de Dados do SEMEAR 28
Figura 13 – Política de Versionamento 29
Figura 14 - Painel de atividades 31
Figura 15 – Estrutura Analítica das Sprints 32
Figura 16 - Diagrama de Implantação 34
Figura 17 - Arquitetura do Sistema 35
Figura 18 - Diretórios gerados ao iniciar uma nova aplicação ReactJs 36
Figura 19 - Diretório src 37
Figura 20 - Funcionamento da arquitetura para cadastro de curso 38
Figura 21 - Arquivo Config/Firebase.js 39
Figura 22 - Arquivo Actions/Course.js: 39
Figura 23 - Importação da Função createCourse() 40
Figura 24 - Chamada da função createCourse 40
Figura 25 - Cadastro Curso 41
Figura 26 - Dados registrados no Firebase 41
Figura 27 - Importação de recursos de autenticação 42
Figura 28 - Método constructor do componente Header 42
Figura 29 - Ciclo de vida do componente Header 43
Figura 30 – Implementação do método de logIn 43
Figura 31 – Implementação do método de signOut 44
Figura 32 - Método renderRedirect 44
Figura 33 – Backlog do Produto 45

LISTA DE ABREVIATURAS E SIGLAS

BDD	*Behavior Driven Development*
GCS	*Gerência de Configuração de Software*
JSX	*JavaScript XML*
P.O.	*Product Owner*
TDIC	*Tecnologias Digitais da Informação e Comunicação*
UI	*User Interface*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO 10

2 FUNDAMENTAÇÃO TEÓRICA 12

2.1 Engenharia de Software 12

2.1.1 Processo de software 12

2.1.2 Levantamento de requisitos 13

2.2 Metodologias Ágeis 13

2.2.1 Kanban 14

2.2.2 Scrum 15

2.2.3 Behavior Driven Development 16

2.2.4 Histórias de usuário e cenários de aceitação 16

2.3 Gerência de configuração 18

2.3.1 Controle de versão 18

2.4 Desenvolvimento do Sistema 18

2.4.1 Javascript 19

2.4.2 React 19

2.4.3 Banco de Dados 23

2.4.4 Bancos NoSql 23

2.4.5 Firebase 24

3 DESENVOLVIMENTO 26

3.1 Processo Metodológico 26

3.1.1 Definição do problema 26

3.1.2 Definição do escopo e levantamento de requisitos 26

3.1.3 Escolha da tecnologia 27

3.1.4 Processo de Software Iterativo Incremental 28

3.1.5 Política de Versionamento 28

3.1.6 Gerenciamento do projeto com os Métodos Kanban e Scrum 30

3.1.7 Planejamento das Sprints 31

3.1.8 Sprint 1 32

3.1.9 Sprint 2 41

3.1.10 Sprint 3 44

4 RESULTADOS ALCANÇADOS 45

5 CONSIDERAÇÔES FINAIS 47

REFERÊNCIAS BIBLIOGRÁFICAS 48

APÊNDICES 51

APÊNDICE A – DOCUMENTO DE VISÃO 51

APÊNDICE B – ESPECIFICAÇÃO DE CASOS DE USO 59

APÊNDICE C – PROTÓTIPOS 62

APÊNDICE D – HISTÓRIAS DE USUÁRIO 72

APÊNDICE E – TELAS DO SISTEMA 84

1 INTRODUÇÃO

Aprender é o processo de assimilação de qualquer tipo de conhecimento, enquanto ensinar pode ser entendido como a ação de fazer com que o outro adquira conhecimento. Segundo Libâneo (1994, p. 90) “a relação entre ensino e aprendizagem não é mecânica, não é uma simples transmissão do professor que ensina para um aluno que aprende.” Ele conclui que “é uma relação recíproca na qual se destacam o papel dirigente do professor e a atividade dos alunos.” Assim, “o ensino visa estimular, dirigir, incentivar, impulsionar o processo de aprendizagem dos alunos”.

Para que o aprendizado aconteça Libâneo (1994) destaca a necessidade de atividades práticas e exercícios como meios de consolidar o estudo e a aplicação prática dos conhecimentos e habilidades obtidos. Pode-se apontar também o processo de avaliação do aprendizado como ferramenta de extrema importância para certificar o conhecimento, pois segundo Libâneo (1994, p. 197) “a correção de erros cometidos possibilita o aprimoramento, a ampliação e o aprofundamento de conhecimentos e habilidades e, desta forma, o desenvolvimento das capacidades cognoscitivas”.

Neste contexto percebe-se que o aprendizado e a construção do conhecimento são viabilizados por meio de processos de comunicação, no entanto, se faz necessário salientar que durante este processo, o aluno deve ser estimulado com conteúdos a seu alcance, como aponta Libâneo (1994), o que por sua vez gera diversos produtos como resultados do estudo tais como trabalhos; resumos e artigos, podendo estes servirem como meio para o aprendizado de outrem.

Dito isso, nota-se que para o desenvolvimento do ensino e aprendizagem faz se necessário atividades de recuperação e comunicação do conhecimento produzido. Filho et al (2012) aponta o surgimento e desenvolvimento da revista científica e outros meios de comunicação em razão da necessidade de troca de experiências e conhecimentos. Contudo, o autor afirma que com o advento das Tecnologias Digitais da Informação e Comunicação (TDIC), novas possibilidades surgiram como as iniciativas de arquivos abertos e movimentos de acesso livre, destacando os repositórios eletrônicos como uma inovação no sistema de comunicação e gerenciamento do conhecimento.

Este estudo tem como objetivo o desenvolvimento de um sistema WEB para a publicação e compartilhamento de trabalhos acadêmicos, sendo este um repositório, no qual

deve auxiliar alunos e professores na disponibilização, acesso e uso destes trabalhos, gerando assim um catálogo de trabalhos onde os mesmos poderão ser lidos, baixados, distribuídos, ou referenciados pelos usuários, podendo também servir de modelo para o desenvolvimento de novos trabalhos, o que permite poupar tempo em pesquisa, melhorar a qualidade e contribuir para a geração do conhecimento (PROSSER, 2004). Vale ressaltar que os trabalhos submetidos no SEMEAR serão moderados por professores ou usuários com privilégios de moderador, nos quais serão definidos pelo administrador.

Outro ponto importante para criação de um repositório acadêmico está na necessidade, por parte das instituições de ensino superior, em estimular a produção discente, uma vez que um dos pontos avaliados pelo Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira (INEP), por meio do Instrumento de Avaliação Institucional Externa (2017), são as políticas institucionais e ações de incentivo a produção discente. Sendo este um dos critérios para a definição do nível de qualidade da Instituição como Faculdade, Centro Universitário ou Universidade.

Em suma, a importância deste trabalho está em maximizar os impactos dos trabalhos realizados na instituição por meio da maximização do seu acesso e uso, contribuindo assim para melhora do processo de ensino-aprendizagem mediante a disponibilização livre dos conteúdos gerados.

No capítulo 2 são apresentados os principais conceitos e temas pertinentes ao trabalho. Em seguida, no capitulo 3 é exposto o desenvolvimento do sistema, evidenciando a escolha das tecnologias; os métodos e ferramentas utilizadas e também o padrão definido para a realização das tarefas. No capítulo 4 são apresentados os resultados alcançados e trabalhos futuros. Por fim, o capitulo 5 descreve os principais aprendizados considerações relativas ao trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo será apresentada a teoria utilizada para o desenvolvimento do presente trabalho, bem como os métodos e ferramentas para a aplicação da engenharia de software, gerenciamento de projetos e tecnologias para o desenvolvimento.

2.1 Engenharia de Software

Segundo Pressman e Maxim (2016, p. 14), “a engenharia de software abrange um processo, um conjunto de métodos (práticas) e um leque de ferramentas que possibilitam aos profissionais desenvolverem software de altíssima qualidade”. Os autores dizem ainda que a engenharia de software é direcionada por um conjunto de princípios que ajudam na produção de softwares eficazes. Sommerville (2011, p. 3) salienta que “quando falamos de engenharia de software, não se trata apenas do programa em si, mas de toda a documentação associada e dados de configurações necessários para fazer esse programa operar corretamente”.

Fornecer valor aos usuários e simplificar são alguns dos princípios que abrangem a engenharia de software (PRESSMAN; MAXIM, 2016). Assim, para este trabalho, a engenharia de software fornecerá processo, métodos, técnicas e ferramentas que contemplarão desde as fases iniciais do projeto, englobando levantamento e análise de requisitos, até as fases finais que se encerra com a utilização do software pelo cliente.

2.1.1 Processo de software

O processo de desenvolvimento de software é um conjunto de atividades a serem seguidas para o desenvolvimento de um software. Segundo Sommerville (2011, p. 18) um processo de software é definido como um “conjunto de atividades relacionadas que levam à produção de um produto de software”. Esse processo é importante “porque propicia estabilidade, controle e organização para uma atividade que pode, sem controle, tornar-se bastante caótica” (PRESSMAN; MAXIM, 2016, p. 52).

Existem diversos modelos de processos, dentre eles o modelo em cascata, incremental, evolucionário. Segundo Pressman e Maxmim (2016) os modelos incrementais oferecem grande vantagem em relação aos lineares, como cascata, permitindo a entrega de funcionalidades ao cliente em menor tempo.

Através da aplicação do modelo incremental em conjunto com a metodologia ágil, é possível entregar valor ao cliente desde os primeiros incrementos. Desse modo, possibilita-se a coleta de *feedback* dos usuário, proporcionando assim, possíveis melhorias nos próximos incrementos.

2.1.2 Levantamento de requisitos

Sommerville (2011, p. 57) define que, "os requisitos de um sistema são as descrições do que o sistema deve fazer, os serviços que ele oferece e as restrições sobre seu funcionamento". Também os classifica em requisitos funcionais, a qual definem as funções que o sistema deve conter, bem como o mesmo deve se comportar em determinadas situações; e em requisitos não-funcionais, relacionados a qualidade que o sistema deve apresentar.

Pressman e Maxmim (2016) classificam a engenharia de requisitos em três etapas: concepção, a qual define a condição e o tamanho do problema a ser resolvido; levantamento, que consiste em um processo de analise para definir o que é necessário; e a elaboração, onde os requisitos básicos são refinados e modificados.

2.2 Metodologias Ágeis

O termo "metodologias ágeis" é utilizado para se referir aos métodos que aplicam o "Manifesto para o desenvolvimento ágil de software", que é um documento assinado por 17 (dezessete) especialistas em processos de desenvolvimento de software, a qual estabelece princípios comuns a serem seguidos (BECK et al., 2001).

Nas metodologias ágeis o foco está na entrega rápida de softwares úteis, para tal, as mesmas prezam por: documentar apenas o necessário, entregas incrementais, respostas rápidas a mudanças, participação ativa do cliente em todo processo, foco no cliente e na entrega de valor contínua. De acordo com Pressman e Maxim (2016, p. 63) as metodologias ágeis "[...] priorizam a entrega mais do que a análise e o projeto (embora essas atividades não sejam desencorajadas); também priorizam a comunicação ativa e continua entre desenvolvedores e clientes.".

Em um processo convencional dirigido a planos, os requisitos são especificados completamente através de uma documentação extensa e detalhada, tornando o processo dispendioso e prolongado, o que acarreta em uma entrega de software final que costuma ser

demorada. Segundo Cohn (2004, p. 258) “o objetivo no desenvolvimento ágil é encontrar o equilíbrio certo entre documentação e discussão”.

2.2.1 Kanban

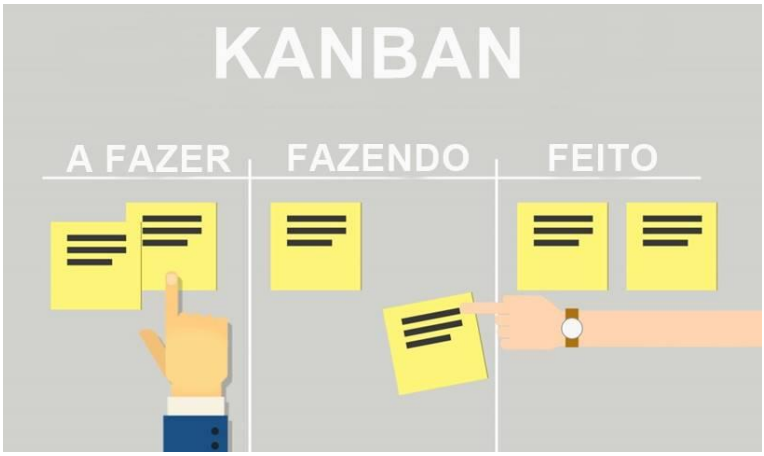
Segundo Egestor (2017) o Kanban é um método que visa aumentar a eficiência da produção e otimizar seus sistemas de movimentação, produção, realização de tarefas e conclusão de demandas. Também conhecido como método de gestão visual.

O sistema Kanban é baseado em referências visuais atreladas aos produtos, lugares comuns, murais de linha de produção ou até mesmo computadores que utilizam método de Kanban eletrônico para funcionar. No Kanban são utilizados cartões (*post-its* na maioria das vezes) com cores e tamanhos diferentes para definir e descrever as tarefas que precisam ser feitas.

Segundo Egestor (2017) o numero de atividades que podem ser feitas simultaneamente devem ser limitados com os *kanbans*. É recomendado limitar estas atividades justamente para evitar o inverso do ócio também experimentado por empresas “a sobrecarga de alguns funcionários em atividades que deveriam ser feitas por mais pessoas, ou que deveriam guardar para que possam ser realizadas com dedicação exclusiva do colaborador responsável”.

O *Kanban* (Figura 1) se baseia principalmente nas colunas “*to do*”, “*doing*” e *“done”,* no entanto existem variações de acordo com características e detalhes específicos de cada linha de produção.

Figura 1 - Ilustração de um quadro Kanban



Fonte: Gradus (2018)

2.2.2 Scrum

O *Scrum* segundo seus desenvolvedores, Schwaber e Sutherland (2017), é um *framework* para tratamento e resolução de problemas complexos e adaptativos de modo criativo e produtivo, objetivando agregar um alto valor ao produto.

O *Scrum* utiliza uma abordagem iterativa e incremental para melhorar a previsibilidade e o controle de riscos. Para isso, Schwaber e Sutherland (2017), definem três pilares no qual o *Scrum* se apoia, sendo eles: a transparência, que todos tenham a mesma visão dos aspectos importantes do processo; a inspeção, verificar os artefatos e o progresso para detectar variações indesejadas; e a adaptação, ajustar aspectos do processo que sofrem desvios além do aceitável para minimizar futuros desvios.

A essência do *Scrum* é um pequeno time de pessoas, constituído por um *Product Owner* (P.O.), o Time de Desenvolvimento e um *Scrum Master* (SCHWABER; SUTHERLAND, 2017). O *Product Owner* (P.O.) é o responsável por gerenciar o *product backlog*; o time de desenvolvimento consiste numa equipe multidisciplinar que é responsável por transformar o *product backlog* em incrementos de funcionalidades possivelmente liberável; e o *Scrum Master* responsável pelo processo e as boas práticas do *Scrum*.

O processo Scrum baseia-se em pequenos ciclos de atividades definidos como Sprint. “O coração do *Scrum* é a *Sprint*, um *time-boxed* de um mês ou menos, durante o qual um ‘Pronto’, incremento de produto potencialmente liberável é criado” (SCHWABER; SUTHERLAND, 2017, p.9).

Para maximizar a transparência, Schwaber e Sutherland (2017), estabeleceram a construção de três artefatos, o *product backlog*, o *sprint backlog* e o incremento. O *product backlog* consiste em uma lista com todos os requisitos conhecidos necessários no produto, mas que, no entanto, é evoluída conforme o produto e o ambiente no qual será utilizado evoluem. O *sprint backlog* é constituído pelos itens do *product backlog* selecionados para a *Sprint*, em conjunto com o plano definido para entregar o incremento do produto e atingir o objetivo da *Sprint*. E o Incremento, que constitui-se de todos os itens do *product backlog* concluídos durante a *Sprint*, juntamente com os incrementos de todas as *Sprints* anteriores.

Para o pleno funcionamento do Scrum, Schwaber e Sutherland (2017) definem quatro eventos formais para manutenção da inspeção e adaptação sendo eles: O “planejamento da

Sprint", que estabelece o trabalho que será realizado na iteração; a "reunião diária", com duração máxima de 15 (quinze) minutos, com o objetivo de definir o que será produzido nas próximas 24 (vinte e quatro) horas e inspecionar o trabalho realizado desde a última reunião; a "revisão da *Sprint*", realizada ao fim de cada *Sprint* no intuito de inspecionar o incremento e ajustar o *backlog* do produto se necessário; e a "retrospectiva da *Sprint*", que oportuniza ao time se autoavaliar para gerar melhorias a serem aplicadas na próxima *Sprint*.

2.2.3 Behavior Driven Development

O *Behavior Driven Development* (BDD) é uma metodologia ágil de desenvolvimento de software orienta pelo comportamento da aplicação, a qual possui as melhores práticas para desenvolvimento de software objetivando entregar o máximo de valor e qualidade no menor prazo. Para tal o BDD promove uma linguagem ubíqua entre clientes e desenvolvedores, que se aproveita como documentação para os requisitos do sistema de modo ativo, por ser menos ambígua (SMART, 2014).

De acordo com Solís e Wang (2011) as especificações de requisitos no BDD são escritas livres de detalhes de implementação e com linguagem orientada ao negócio. Para isso as mesmas são descritas através de Histórias de Usuário e Cenários de Aceitação, as quais buscam apresentar o comportamento do sistema do ponto de vista do usuário.

Smart (2014) apresentou em seu livro os seguintes possíveis benefícios do BDD: redução de gastos, concentrando o esforço do desenvolvimento na descoberta e entrega de funcionalidades que proverão valor ao negócio; redução dos custos, como consequência da redução dos gastos; mudança fáceis e seguras, como resultado das especificações executáveis que utilizam termos que os *stakeholders* estão familiarizados; e, releases mais rápidas, por causa que testes automatizados também aceleram o ciclo de release consideravelmente.

2.2.4 Histórias de usuário e cenários de aceitação

A aplicação do BDD se inicia com a construção das Histórias de Usuário e a descrição dos Cenários de Aceitação as quais, Pressman e Maxim (2016) afirmam que devem descrever o resultado, características e funcionalidades a serem implementadas, além disso, segundo Prikladnicki, Willi, Milani (2014) as histórias de usuários não devem conter detalhes de implementação, no entanto, têm de conseguir transmitir as necessidades do negócio.

Uma boa história de usuário deve ser independente, negociável, possuir valor, ser estimável, pequena e testável de acordo com Cohn (2004). Ele afirma que a dependência entre as histórias de usuário pode causar problemas de priorização e estimativa, portanto, devem ser independentes e negociáveis, pois, não são imutáveis, têm de ser formuladas em consenso entre os clientes e a equipe de desenvolvimento; devem possuir valor, pois, em muitos projetos existem funcionalidades que não possuem utilidade para o negócio; devem ser estimáveis, pois, os desenvolvedores as transformarão em código executável; devem ser pequenas, tendo em vista que várias histórias pequenas comporão histórias grandes; e por fim, testáveis, para que quando os testes forem executados, os mesmos provem que a funcionalidade foi implementada com sucesso.

Prikladnicki, Willi, Milani (2014, p. 44) apresenta um formato que normalmente é utilizado para descrever histórias de usuário: "Como um **<tipo_de_usuário>**, eu gostaria de **<funcionalidade>** para **<benefício>**". A Figura 2 abaixo reproduz um exemplo de história de usuário, onde percebe-se que o requisito é descrito em linguagem natural, facilitando assim o entendimento de ambas as partes.

Figura 2 - Exemplo de uma História de Usuário

HISTÓRIA DE USUÁRIO	DESCRIÇÃO	STATUS	PRIORIDADE
UH. 002	Manter Cadastro	Em desenvolvimento	Essencial
Eu como: Usuário.			
Quero: Realizar o cadastro de minhas informações no sistema			
Para: Que eu possa ter acesso as informações e funcionalidades presentes no sistema.			
CRITÉRIOS DE ACEITAÇÃO			
ITEM	DETALHAMENTO		
1	O sistema apresenta a tela "Inicial".		
2	Usuário informa corretamente todos os dados do formulário.		
3	Usuário aciona o botão "Criar Conta".		
3.1	Redireciona para a tela "Lista de Cursos".		

Fonte: Os Autores

Definida a história de usuário, segue-se para a elaboração dos critérios de aceitação, também representados na Figura 2, as quais descrevem eventos que ocorrem no sistema a partir de determinadas ações, representando assim comportamentos esperados. (HELM; WILDT, 2014).

2.3 Gerência de configuração

A Primeira Lei da Engenharia de Sistemas diz que: “Não importa onde você esteja no ciclo de vida do sistema, o sistema mudará, e o desejo de alterá-lo persistirá por todo o ciclo de vida” (PRESSMAN; MAXIM, 2016, p. 624).

A manutenção do sistema é uma tarefa prevista dentro das metodologias ágeis. Devido às características iterativa e incremental das mesmas, presume-se que ao longo do processo de desenvolvimento de uma aplicação diversas modificações serão realizadas, originando-se a necessidade de controla-las. Desse modo surge a Gerência de Configuração de Software (GCS).

Pressman e Maxim (2016) caracterizam a GCS como um conjunto de tarefas destinadas a controlar as alterações, sendo responsável por identificar e relacionar os artefatos que necessitam ser alterados; definir mecanismos para gerenciar diferentes versões dos mesmos; auditar e relatar as alterações feitas.

2.3.1 Controle de versão

O controle de versão é uma atividade indispensável no processo de desenvolvimento de qualquer aplicação. “O controle de versão é um sistema que registra as mudanças feitas em um arquivo ou um conjunto de arquivos ao longo do tempo de forma que você possa recuperar versões específicas” (GIT). Humble e Farley (2014) salientam que qualquer artefato relacionado ao sistema (como o código fonte; scripts de bancos de dados; documentação; entre outros), pode e deve estar sob o controle de versão.

Atualmente existem diversos softwares de controle de versão, dentre os quais, destaca-se o Git que é um sistema *open source*, que opera de forma totalmente distribuída; possui um design simples; suporta um desenvolvimento não-linear e é capaz de lidar de modo eficaz com grandes projetos. Concebido por Linus Torvald no ano de 2005, caracteriza-se por ser fácil de usar, leve e rápido (GIT).

2.4 Desenvolvimento do Sistema

O processo de construção de um produto de software envolve diversas atividades como modelagem de banco de dados, modelagem de requisitos, modelagem de processos, testes, entre outras. Também é indispensável o uso de linguagens de programação e

ferramentas que suportem o desenvolvimento. A seguir são explicitadas algumas tecnologias que são utilizadas para o desenvolvimento de aplicações.

2.4.1 Javascript

Segundo o site ecma-internacional.org (2018) desde a publicação da primeira versão da linguagem, em 1997, a *ECMAScript* ou *Javascript* popularmente conhecido, tornou-se uma das linguagens de programação de uso geral mais utilizadas no mundo. Algum tempo atrás seu uso era mais comum apenas em navegadores da Web, mas atualmente foi amplamente adotada para aplicativos incorporados e de servidor.

Como mencionado o Javascript era mais usado incialmente como um complemento nos navegadores, entretanto hoje seu uso vai muito além do seu propósito inicial. Existem diversas tecnologias no mercado que aumentam ainda mais as possibilidades de uso da linguagem. Não tem como falar da linguagem sem mencionar o *NodeJS*. Segundo o site node.org, o *NodeJs* é um interpretador javascript baseado no *Chorme's V8 Javascript Engine*, que é o responsável por possibilitar executar a linguagem no servidor.

Outros recursos disponível para a utilização do Javascript, é o *npm* (*node package manager*). Com ele é possível gerenciar as dependências dos projetos. Como exemplo, caso queira executar o *JQuery* em um projeto utilizando o *npm*, basta executar o comando, *$npm install jquery* para que o *donwload* seja realizado.

Os exemplos citados acima são apenas uma pequena parte do que pode ser feito com Javascript. Atualmente a linguagem é amplamente utilizada em testes automatizados, aplicações desktop, aplicações server, frameworks para interfaces em sistemas web, entre outros. O que permite que um ecossistema de desenvolvimento possa ser cem por cento desenvolvido em *Javascript*. A vantagem de utilizar uma única linguagem em diferentes áreas de um projeto, é possibilitar que uma pessoa com conhecimento bem acentuados em uma linguagem possa experimentar novas áreas, ou que uma pequena empresa possa reaproveitar os conhecimentos da equipe (HANASHIRO, 2018)

2.4.2 React

O *React* não é um framework, mas sim "uma biblioteca Javascript declarativa, eficiente e flexível para criar interfaces visuais", desse modo, o *React* serve para criar

interfaces visuais tendo como principio o desenvolvimento por componentes. Portanto, por definição, ele tem apenas um concerne: renderizar a interface de usuário (UI.) (LIMA, 2017).

Apesar do *React* ser uma biblioteca para o desenvolvimento de interfaces Lima (2017) explica que o mesmo possui todo um ecossistema que o deixa mais robusto e completo. Devido a isso ele afirma que é possível criar aplicações completas usando o React. Este ecossistema é formado por: *React;* JSX*;* ES2015*; Webpack; Flux/Redux; Axios/fetch* e *Jest/Mocha.*

O JSX é uma sintaxe semelhante ao XML, na qual é possível compreender de uma forma melhor como será montado o seu componente na UI. Na documentação do *React* (2019) diz que a vantagem de utilizar o JSX é porque o *React* adota o fato de que a lógica de renderização é inerentemente associada a outra lógica de interface de usuário. Como os eventos são tratados e como o estado muda com o tempo. Na documentação do *React* (2019) consta que ao invés de separar artificialmente as tecnologias, colocando marcação e lógica em arquivos separados, o *React* separa as preocupações com unidades fracamente acopladas chamadas de componentes.

O *React* é baseado nestes componentes, para que seja possível maximizar o reuso na construção da aplicação. Em resumo, os componentes em *React* deixam o desenvolvimento mais produtivo. Com eles temos um reaproveitamento de código maior e uma menor probabilidade de novos bugs serem introduzidos na aplicação (LIMA, 2017). A Figura 3 a seguir demonstra um exemplo de componente de botão em *React*:

Figura 3 - Componente em React

```
1  import React from 'react';
2
3  export default function Button() {
4    return (
5      <button
6        onClick={() => alert('Clicked')}
7      >Click me</button>
8    );
9  }
```

Fonte: Lima (2017)

Em todos os tipos de paradigmas no desenvolvimento de software, passar parâmetros é extremamente comum. Com os componentes do *React* isso não é diferente. A diferença é que

no *React*, utilizam-se os *props* (abreviação para *properties*). Nas imagens a baixo é possível ver a diferença dos componentes com e sem *props*:

Na Figura 4 abaixo o React retorna na tela o texto “Hello World” de forma estática.

Figura 4 - Componente em React sem props.

```
1  import React from 'react';
2
3  export default function HelloWorld() {
4    return (
5      <h1>Hello World</h1>
6    );
7  }
```

Fonte: Lima (2017)

Já na Figura 5 a seguir é possível alterar de forma dinâmica, a propriedade do componente que será renderizado na tela do usuário através da propriedade *props.name*.

Figura 5 - Componente em React com props.

```
1  import React from 'react';
2
3  export default function HelloWithProps(props) {
4    return (
5      <h1>Hello {props.name}</h1>
6    );
7  }
```

Fonte: Lima (2017)

Outra característica importante do *React* é o *State*. O estado (ou estado) da aplicação pode ser definido como: o lugar onde os dados vem e se transformam ao longo do tempo. Dito isso, os componentes *React* podem ser divididos em duas categorias: *Presentational* e *Container*. Os componentes do tipo *Presentational*, se importam somente com a apresentação dos dados já os componentes do tipo Container, além da apresentação dos dados, tem que lidar também com algum tipo de lógica, ou transformação de dados (LIMA, 2017).

O tipo *Presentational* não trabalha com estados, se importando apenas na apresentação dos dados. Abaixo pode ser visto um exemplo de componente *presentational*:

Figura 6 - Exemplo de componente *presentational*

```
1  import React from 'react';
2
3  export default function Presentational() {
4    return (
5      <h1>Hello World</h1>
6    );
7  }
```

Fonte: Lima (2017)

Já os componentes do tipo *container*, além de apresentar os dados, tem que tratar de algum tipo de lógica ou transformação do dado. Na Figura 7, por exemplo, mostra o estado do componente recebendo ‘World’ dentro do método construtor:

Figura 7 - Exemplo do estado do componente recebendo ‘World’

```
1  import React from 'react';
2
3  export default class Container extends React.Component {
4    constructor(props) {
5      super(props);
6      this.state = { name: 'World' };
7    }
8
9    render() {
10     return (
11       <h1>Hello {this.state.name}</h1>
12     );
13   }
14 }
```

Fonte: Lima (2017)

Uma das grandes vantagens da utilização do *React* é a maneira que realiza a manipulação do DOM. A manipulação do DOM é mais lenta do que a maioria das operações

feitas apenas no *javascript*. Por este motivo o *Facebook* criou o *VirutalDOM*. O *VirtualDOM* faz uma cópia de todos os nós em código *Javascript*. Deste modo somente quando necessário as alterações são repassadas para o DOM original (LIMA, 2017).

Outro recurso disponibilizado para instalação via *NPM* para a criação de projetos em ReactJs, é o ambiente Create React App . Segundo a própria documentação oficial do *React* o Create React App é um ambiente confortável para aprender React**.** É a melhor maneira de começar um *single-page application* em React. Já que define um estrutura inicial e funcional para a base do projeto no *frontend*.

Para iniciar um projeto React utilizando este ambiente basta executar o comando *npm* create-react-app nome_do_projeto no terminal ou CMD. Em seguida necessário entrar no diretório do projeto criado e excecutar o comando *npm* start para iniciar todas as dependencias necessárias para o funcionamento do projeto.

2.4.3 Banco de Dados

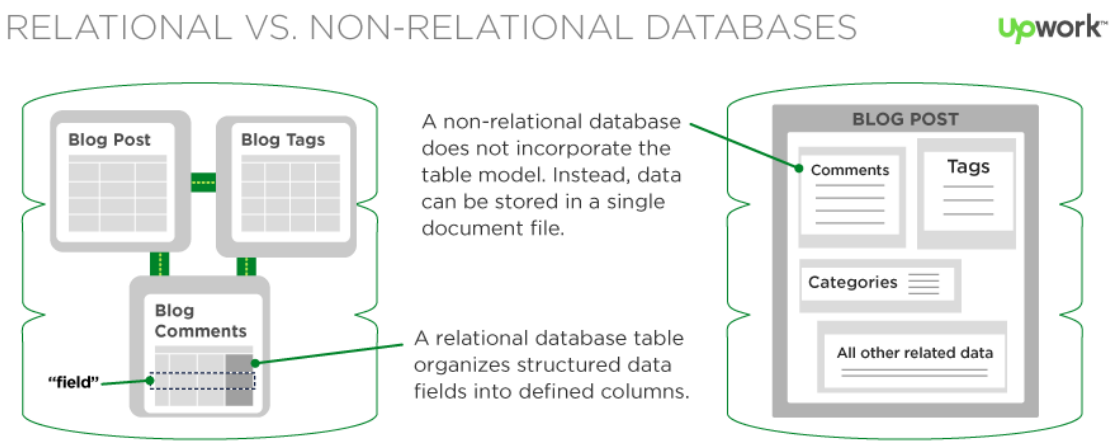
Os bancos de dados relacionais têm sido a escolha padrão para o armazenamento de grandes volumes de dados, principalmente no meio corporativo. Após um longo período o interesse em banco de dados NoSQL está aumentando gradativamente. Este fato se dá pelo crescimento massivo das informações geradas em grandes corporações, e a necessidade de trabalhar com quantidades gigantescas de dados, sendo esta uma alternativa aos bancos de dados relacionais (SADALAGE, 2013).

2.4.4 Bancos NoSql

A maior motivação para utilizar banco de dados não relacionais NoSQL é resolver o problema de escalabilidade presente nos bancos de dados tradicionais. Já que uma das maiores dificuldades dos bancos relacionais é o alto custo e a complexidade para escalar com eficiência um banco que utiliza SQL. Atualmente um dos grandes desafios na área de computação é a manipulação e processamento de grande quantidade de dados no contexto de Big Data. O conceito *Big Data* pode ser resumidamente definido como uma coleção de bases de dados tão complexa e volumosa que se torna muito difícil (ou impossível) e complexa fazer algumas operações simples (e.g., remoção, ordenação, sumarização) de forma eficiente utilizando Sistemas Gerenciadores de Bases de Dados (SGBD) tradicionais (VIEIRA et al, 2012)

Apesar de muitos acharem que a facilidade da escalabilidade no banco se dá pelo fato da ausência de *schema*, já que esta estrutura tem como função verificar a integridade das informações e de relacionamento dos dados, o principal motivo é que bancos de dados NoSQL trabalham com clusters, com isso cada nó se comunica com outros através de um sistema *peer-to-peer* gerando redundância das informações em cada partição (STEPPAT, 2009). Outra característica que distingui os dois tipos de bancos, é que, como já dito, no relacional as informações possuem um relacionamento através de tabelas, já no não relacional toda a informação é guardada dentro de um único documento, como sugere a Figura 8 abaixo:

Figura 8 - Banco de dados Relacional versus Não Relacional



Fonte: WODEHOUSE

2.4.5 Firebase

Deste modo possuindo as vantagens de um banco não relacional, atualmente se encontra no mercado o *Firebase* que segundo o Google é uma suíte de desenvolvimento *mobile/web* com infinitas possibilidades. Ele permite criar aplicações sem necessidade de servidor, ou ainda, enviar push *notifications* no smartphone de usuários de uma aplicação de maneira simples, segura e escalável.

Segundo o Google, com o Firebase pode-se criar aplicativos rapidamente, sem a necessidade de gerencias a infraestrutura, além disso, ele oferece funcionalidades como análises, bancos de dados (Cound Firestore DB), serviço de armazenamento de arquivos e imagens (Cloud Storage), *Authentication* para gerenciamento de autenticação de usuários de modo personalizado ou através de outras redes sociais e relatórios de falhas, o que possibilita adaptar-se rapidamente e se concentrar nos usuários. O Firebase oferece uma gama bem

extensa de serviços para o desenvolvimento de aplicações. Um dos diferenciais do Firebase é que além do banco de dados, é possível hospedar uma aplicação web utilizando SSL no próprio servidor.

A integração com o *Firebase* é bem simples. Para a conexão com um *webapp*, por exemplo, basta colar algumas linhas Javascript para que a conexão já esteja configurada. A Figura 9 abaixo mostra um exemplo de conexão com o banco de dados do Firebase:

Figura 9 - Conexão com banco do Firebase

```
<script src="https://www.gstatic.com/firebasejs/5.9.1/firebase.js"></script>
<script>
  // Initialize Firebase
  // TODO: Replace with your project's customized code snippet
  var config = {
    apiKey: "<API_KEY>",
    authDomain: "<PROJECT_ID>.firebaseapp.com",
    databaseURL: "https://<DATABASE_NAME>.firebaseio.com",
    projectId: "<PROJECT_ID>",
    storageBucket: "<BUCKET>.appspot.com",
    messagingSenderId: "<SENDER_ID>",
  };
  firebase.initializeApp(config);
</script>
```

Fonte: Firebase

O *Fireabase* utiliza diversas linguagens para manipulação dos dados entre elas estão: *C++*, *Java*, *Go*, *Javascript*. Tudo depende da tecnologia e ambiente em que está desenvolvendo. Para aplicações *web* pode ser utilizado o *Javascript* para a execução das instruções. A Figura 10, abaixo, demonstra a utilização do método feito em *Javascript* para retornar do banco de dados a quantidade de estrelas de uma postagem:

Figura 10 - Método que retornar a quantidade de estrelas de uma postagem

```
var starCountRef = firebase.database().ref('posts/' + postId + '/starCount');
starCountRef.on('value', function(snapshot) {
  updateStarCount(postElement, snapshot.val());
});
```

Fonte: Firebase

3 DESENVOLVIMENTO

Esse capítulo apresentará os passos realizados para o desenvolvimento do software utilizando o modelo interativo incremental integrado com o *framework Scrum* juntamente com o método *Kanban*. Será apresentado o processo de desenvolvimento que inclui o processo de controle de versão do código, a arquitetura desenvolvida e os padrões de implementação dos requisitos.

3.1 Processo Metodológico

O método qualitativo será o método para o desenvolvimento deste trabalho, apoiando-se em técnicas de coleta de dados, também qualitativas. De acordo com Neves (1996, p.01), a pesquisa qualitativa não busca enumerar ou medir eventos. Ela serve para obter dados descritivos que expressam os sentidos dos fenômenos.

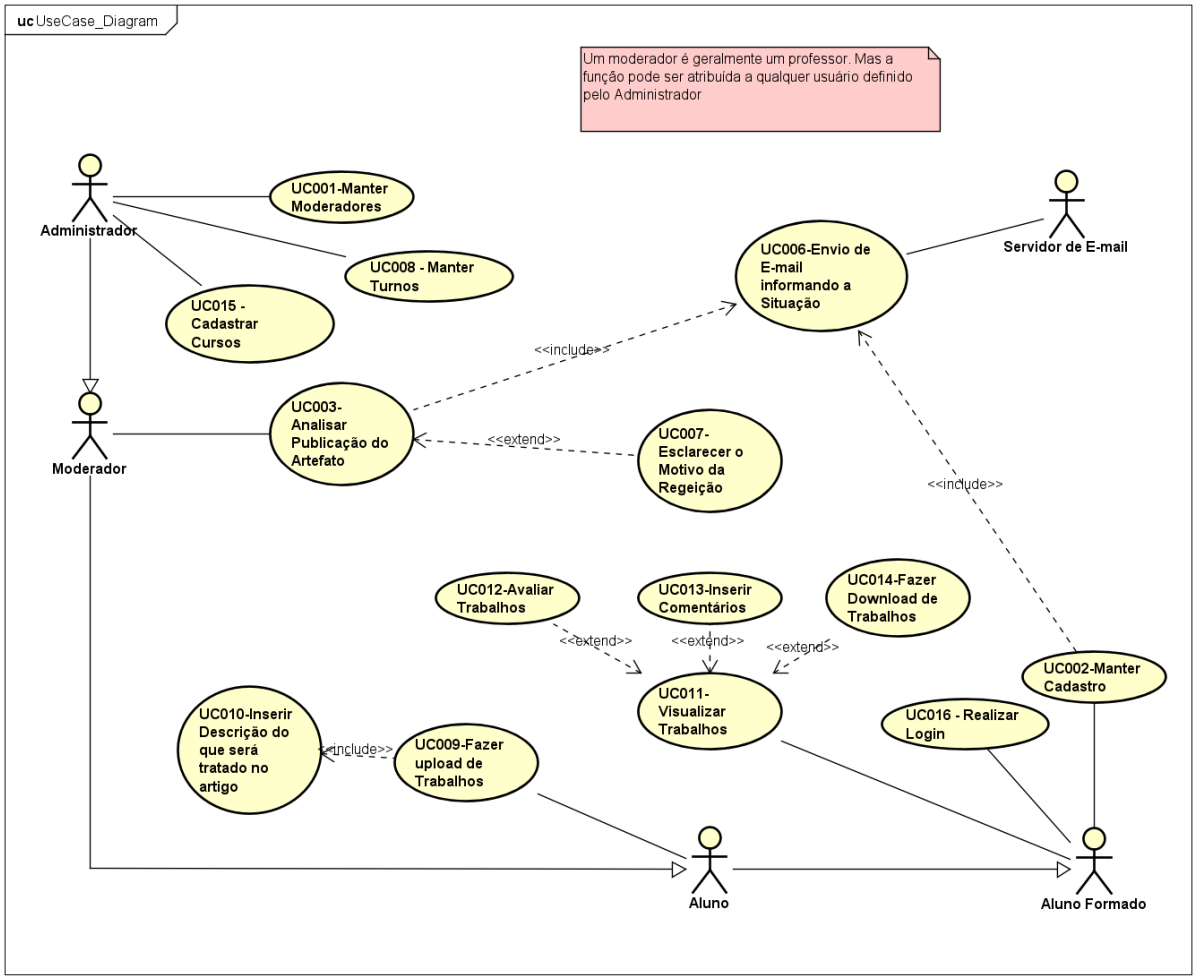
3.1.1 Definição do problema

Inicialmente o problema foi escolhido devido à observação feita durante a disciplina de Projeto Interdisciplinar V do curso de Engenharia de Computação. Foi levantada a ideia de desenvolver uma plataforma de publicação dos trabalhos desenvolvidos por alunos do curso, já que a maioria destes trabalhos ficam inacessíveis após a apresentação das atividades ou conclusão do período de aulas. Deste modo, foi proposto o desenvolvimento do SEMEAR, um sistema que possibilita interações e publicação de trabalhos acadêmicos desenvolvidos por alunos dos cursos oferecidos pela instituição.

3.1.2 Definição do escopo e levantamento de requisitos

Foi levantado com alguns professores do curso de Engenharia de Computação, a definição do escopo e as principais funcionalidades que o sistema deveria possuir. Foram incluídos inicialmente no documento de visão (APÊNDICE A) os principais requisitos do sistema, que são: Manter Usuário, Manter Artefatos, Autenticação e Notificação. Partindo destes requisitos foi elaborado o diagrama de casos de uso (Figura 11) onde tivemos a identificação dos demais requisitos. Com os casos de uso definidos o próximo passo foi realizar a especificação dos mesmos (APÊNDICE B).

Figura 11 - Diagrama de Casos de Uso



Fonte: Os Autores

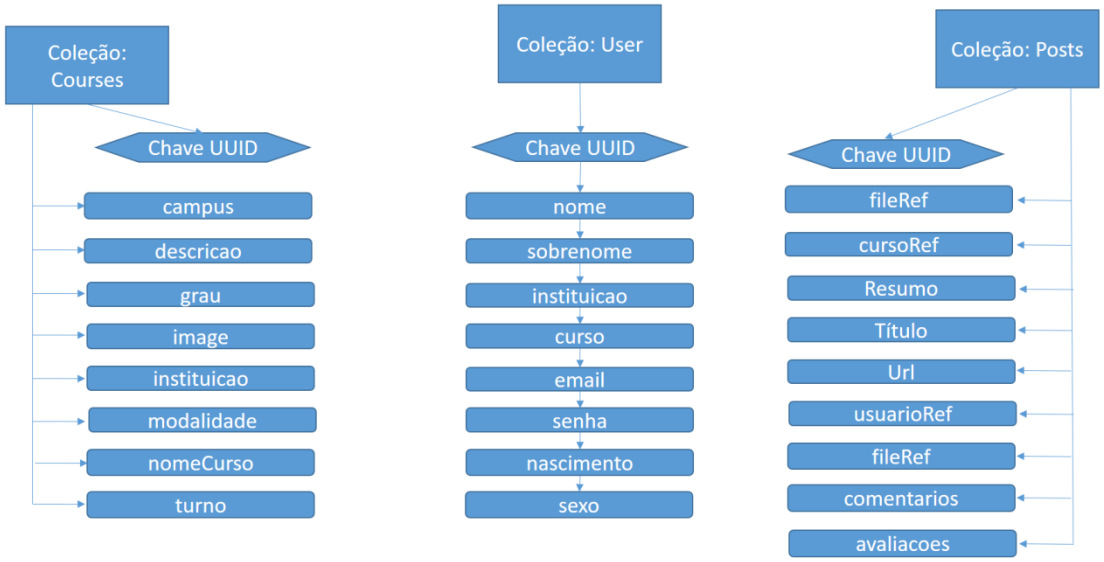
3.1.3 Escolha da tecnologia

Os protótipos (APÊNDICE C) foram desenvolvidos ainda durante o projeto interdisciplinar V ocorrido no primeiro semestre de 2017, utilizando a linguagem Java juntamente com o framework Spring boot. Inicialmente as tecnologias escolhidas para o desenvolvimento foram definidas baseadas no modelo arquitetural MVC (*Model View Controler*), pelo motivo de ser a arquitetura que o time tinha maior conhecimento técnico na época.

Com o surgimento de novas tecnologias e o interesse do time de desenvolvimento em utilizar novas opções disponíveis no mercado, foi decidido à utilização de um banco não relacional, visto que essa tecnologia atendia o propósito do projeto. A Figura 12 representa

como foi definida a estrutura do banco de dados do SEMEAR utilizando um banco não relacional.

Figura 12 – Estrutura do Banco de Dados do SEMEAR



Fonte: Os Autores

Para consolidar esta decisão as tecnologias escolhidas como ferramentas de desenvolvimento foram o *ReactJS* e o *Firebase* (tópicos 2.4.2 à 2.4.5), considerando os interesses do time e também os requisitos, uma vez que estas tecnologias atendem muito bem a necessidade do produto.

3.1.4 Processo de Software Iterativo Incremental

Conforme descrito no tópico 2.1.1 o processo de software é constituído por um conjunto de tarefas relacionadas que resultam na produção de um software. Dentre os diversos modelos de processos existentes foi escolhido o modelo iterativo incremental, pois, o mesmo permite a entrega continua de funcionalidades desenvolvidas possibilitando assim, conferir valor desde os primeiros incrementos, e deste modo, proporcionando também a coleta de *feedbacks* que servem para a validação dos requisitos e definição de possíveis melhorias.

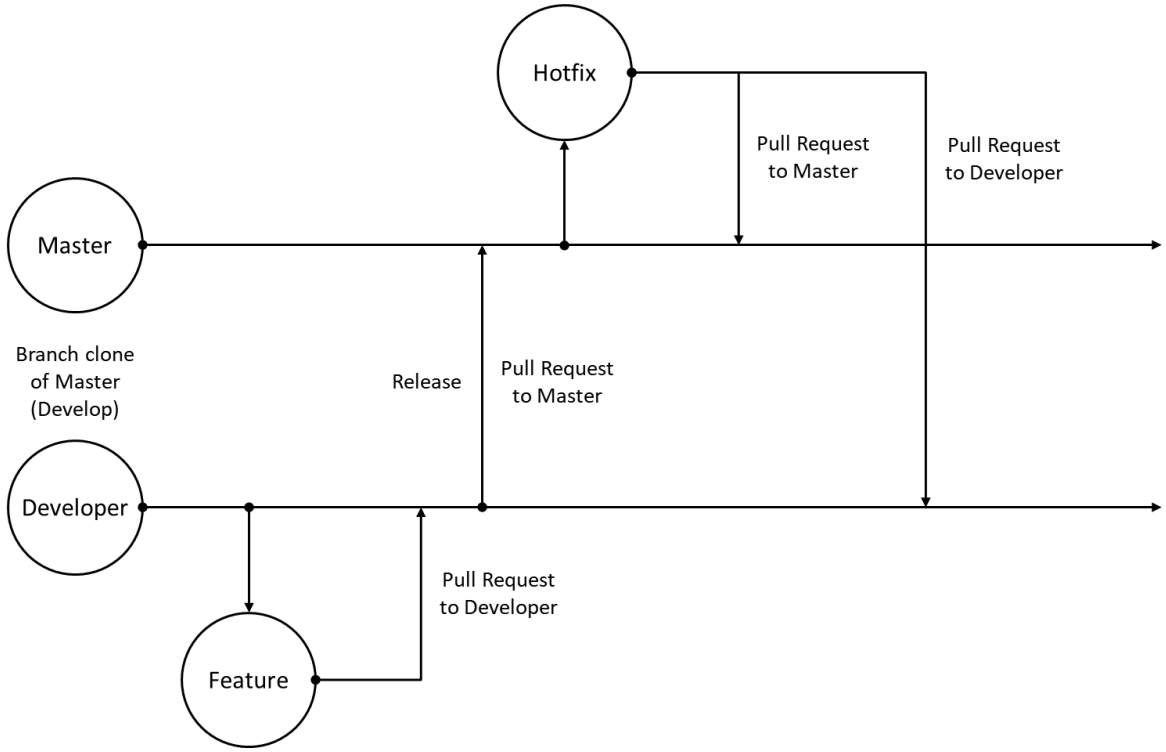
3.1.5 Política de Versionamento

Antes de iniciar o desenvolvimento ficou decidido pela equipe que o controle de versão dos artefatos produzidos será feito através da utilização do sistema Git (tópico 2.3.1),

pois, o mesmo é gratuito e de fácil utilização além de suportar desenvolvimento não-linear através da fácil criação de *branches*, o que permite diferentes abordagens no processo de desenvolvimento de um software.

Com o software de versionamento definido, o próximo passo foi criar um modelo para padronizar o gerenciamento de versões do código, a fim de, auxiliar a equipe de desenvolvimento a evitar problemas como: conflitos de código, versões duplicadas e/ou desatualizadas e a liberação do código. O modelo concebido está descrito na Figura 13, na qual ilustra cada elemento da politica de versionamento.

Figura 13 – Política de Versionamento



Fonte: Os Autores

Como observa-se na Figura 13, a politica de versionamento do projeto baseia-se em duas *branches* a *Master* e a *Developer*, nas quais têm a mesma duração do ciclo de vida do projeto, e duas *branches* auxiliares *Feature* e *Hotfix*, que têm um ciclo de vida mais curto. A *branch* principal denominada *Master* conterá todo o código estável do sistema, enquanto a *branch* secundária denominada *Developer* armazenará o código em desenvolvimento.

Na seleção de atividades, foi definido que o responsável por uma atividade antes de inicia-la deve criar uma nova *branch* auxiliar (*Feature)* com base na *Developer* e implementar

toda a solução da atividade nesta *branch*. Após o desenvolvimento da atividade o responsável deve realizar a junção (*merge*) da mesma com a *branch Developer* e remover a *branch* auxiliar (*Feature*). Posteriormente, pode ser feito a liberação do código para a *branch Master* por meio de *merge*.

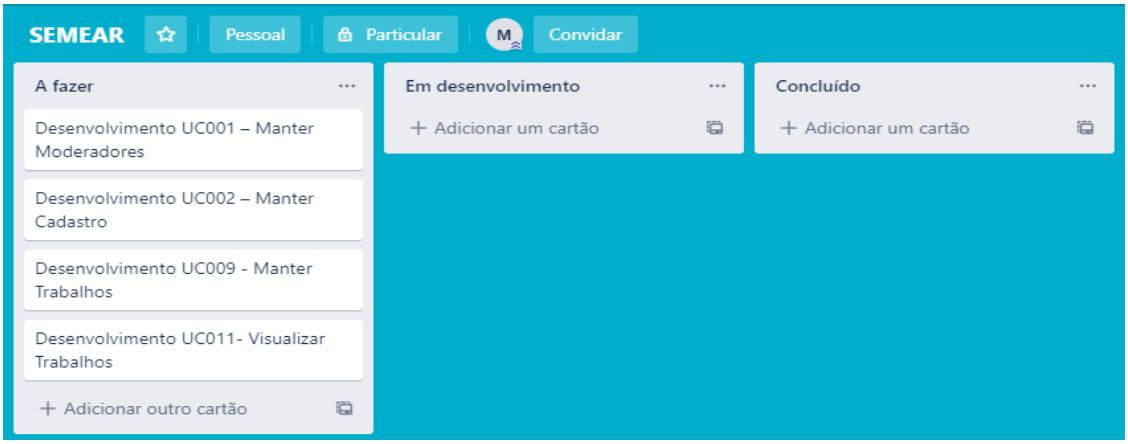
Durante o uso do sistema, caso seja descoberto algum erro ou falha no mesmo, deve-se criar uma nova *branch* denominada *Hotfix* a partir do conteúdo da *Master*. A *branch Hotfix* é criada com o objetivo de realizar correções de problemas que só foram detectados na versão principal do sistema. Após a correção deve-se realizar a junção da mesma com a *branch Master* e também a *Developer* para que o problema não apareça novamente em versões futuras do sistema.

Desse modo conseguiu-se padronizar o versionamento do sistema, separando o código em desenvolvimento do código estável, proporcionando maior segurança durante o processo e evitando o envio de funcionalidades não finalizadas para versão estável do sistema. Além disso, foi possível separar individualmente as atividades em andamento, evitando assim conflitos a todo instante.

3.1.6 Gerenciamento do projeto com os Métodos Kanban e Scrum

Com a política de versionamento definida seguiu-se em estabelecer um modelo para gerenciamento de todas as atividades a serem realizadas durante construção do projeto. Para tal foi decidido que seria utilizado o método *Kanban* (tópico 2.2.1) que é baseado em referências visuais que descrevem o andamento das tarefas. Essa escolha foi feita devido à equipe ser constituída por apenas duas pessoas, sendo assim, uma abordagem mais simplificada tornaria o processo mais eficiente. Neste contexto a escolha do *Kanban* deu-se por ser um método eficiente, porém simples. A Figura 14 descreve o painel de atividades desenvolvido baseado no método *Kanban*.

Figura 14 - Painel de atividades



Fonte: Os Autores

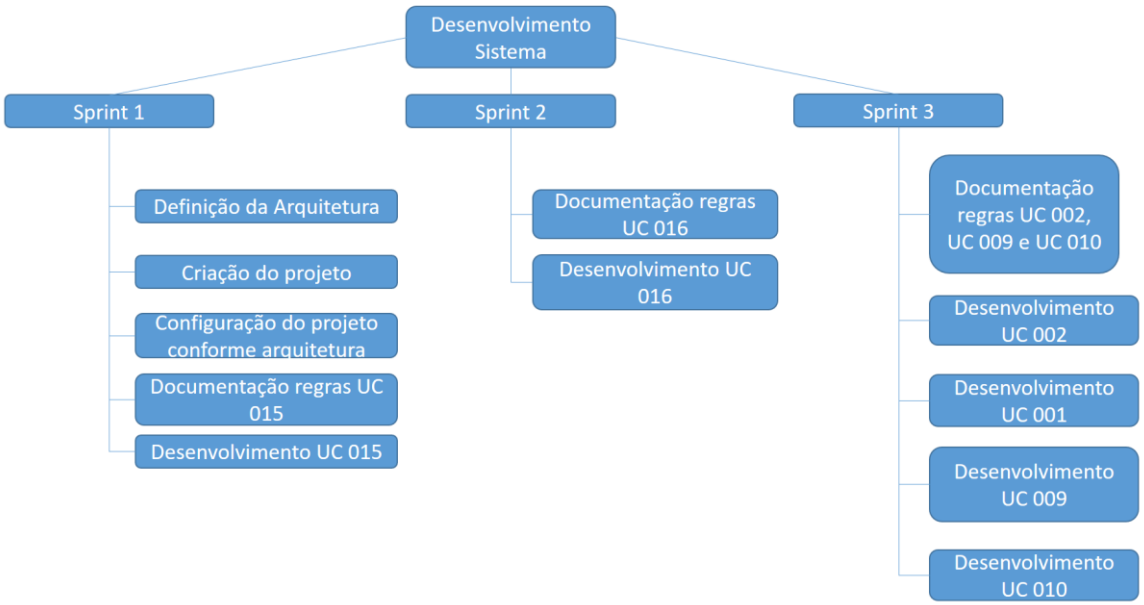
Para complementar o método *Kanban* foi utilizado conceitos oriundos do *framework Scrum* (tópico 0) que também é um método ágil. Foram empregados os conceitos de *Sprint* e Incrementos compreendendo o planejamento das *Sprints* e a elaboração do *backlog* das mesmas.

Essa abordagem permitiu maior clareza do andamento do projeto, através da utilização do *Kanban*, ao mesmo tempo em que possibilitou um melhor controle sobre a prioridade e prazo de cada atividade, por meio *Scrum*.

3.1.7 Planejamento das Sprints

Com a aplicação do *Scrum* foi possível planejar as *sprints* selecionando as atividades de acordo com a prioridade dos requisitos e posteriormente anexa-las ao *Kanban* do projeto. Desse modo obteve-se mais organização durante o decorrer do processo e controle sobre as atividades a serem desenvolvidas. A Figura 15 contém a estrutura analítica das *sprints* executadas onde é possível visualizar o *backlog* de cada *sprint*.

Figura 15 – Estrutura Analítica das Sprints



Fonte: Os Autores

3.1.8 Sprint 1

3.1.8.1 Construção da Arquitetura e Desenvolvimento

A elaboração da arquitetura do sistema seguirá o padrão das tecnologias escolhidas para o projeto. As estimativas e a definição dos ciclos de desenvolvimento foram baseadas nos métodos *Kanban* e *Scrum*. O desenvolvimento do projeto seguirá a política de versionamento definida a fim de garantir a integridade e consistências dos artefatos desenvolvidos.

Os casos de teste serão elaborados e executados com base nos principais requisitos da aplicação a ser desenvolvida, visando simular da forma mais fiel possível o uso do sistema. Serão aplicados testes de usabilidade, carga, interface e testes de integração, a fim de garantir que ele funcionará corretamente quando submetido à utilização real.

Por fim, a pesquisa se caracteriza como experimental, observando os critérios definidos. O resultado da pesquisa consistirá em entregar uma prova de conceito (POC), que possa auxiliar os discentes e docentes na avaliação e publicação dos trabalhos desenvolvidos, e principalmente aos alunos como meio de busca para o desenvolvimento de novos trabalhos.

A fim de aplicar as metodologias definidas por autores com vasto conhecimento sobre o processo de desenvolvimento de software.

3.1.8.2 Requisito modelo e Arquitetura do sistema

No desenvolvimento da sprint 1, foi definido o requisito modelo para o inicio do desenvolvimento do sistema. O requisito escolhido como base, foi o cadastro de cursos, no qual têm características semelhantes a maioria dos requisitos presentes na prova de conceito do sistema. Como mencionado no tópico 2.1.2 Levantamento de requisitos, a técnica utilizada para especificação dos requisitos, foi a elaboração de Histórias de Usuário estando as mesmas disponíveis no APÊNDICE – D.

Arquitetura é outra maneira de ver a implementação, analisando e compreendendo como uma mudança em determinado ponto do sistema afeta o sistema inteiro, e de todas as maneiras possíveis (SILVEIRA et al, 2011).

Dominar uma plataforma, seus principais frameworks e seu funcionamento interno é essencial para que os responsáveis pelo desenvolvimento de uma aplicação conheçam não só as vantagens, mas também as desvantagens de suas decisões. O desenvolvedor que acredita não existir ou desconhece o lado negativo para determinada abordagem sofrerá as consequências de manutenção quando as mesmas surgirem dentro da aplicação (SILVEIRA et al, 2011).

Baseando-se em algumas premissas citadas por Silveira et al (2011), foram levados em consideração três pontos principais para o design e tecnologias escolhidas para atender a necessidade do sistema. São eles:

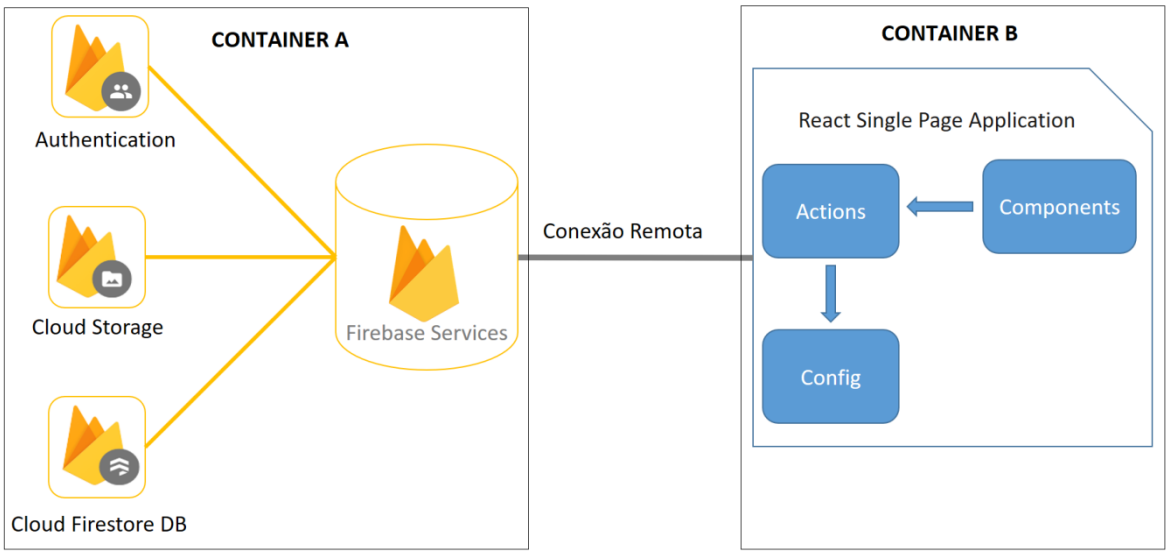
- Uso de tecnologias que estão sendo utilizadas no mercado e que possuem comunidades ativas para suporte;
- Agilidade no desenvolvimento e alto nível de reaproveitamento;
- Iniciar o projeto com baixo custo e possibilidade de escalonamento de acordo com que o produto evolui.

O ReactJs disponibiliza uma arquitetura de pastas quando iniciado através do comando *create-react-app* no *NPM* como demonstrado no tópico 2.4.2 React. Entretanto a biblioteca não define uma arquitetura padrão para ser adotada. Neste caso fica sob

responsabilidade da equipe de desenvolvimento definir a melhor arquitetura que atenda ao projeto.

Na Figura 16 abaixo o diagrama de implantação mostra uma visão geral dos módulos do sistema divididos em duas partes principais. A aplicação React e o Firebase Services:

Figura 16 - Diagrama de Implantação

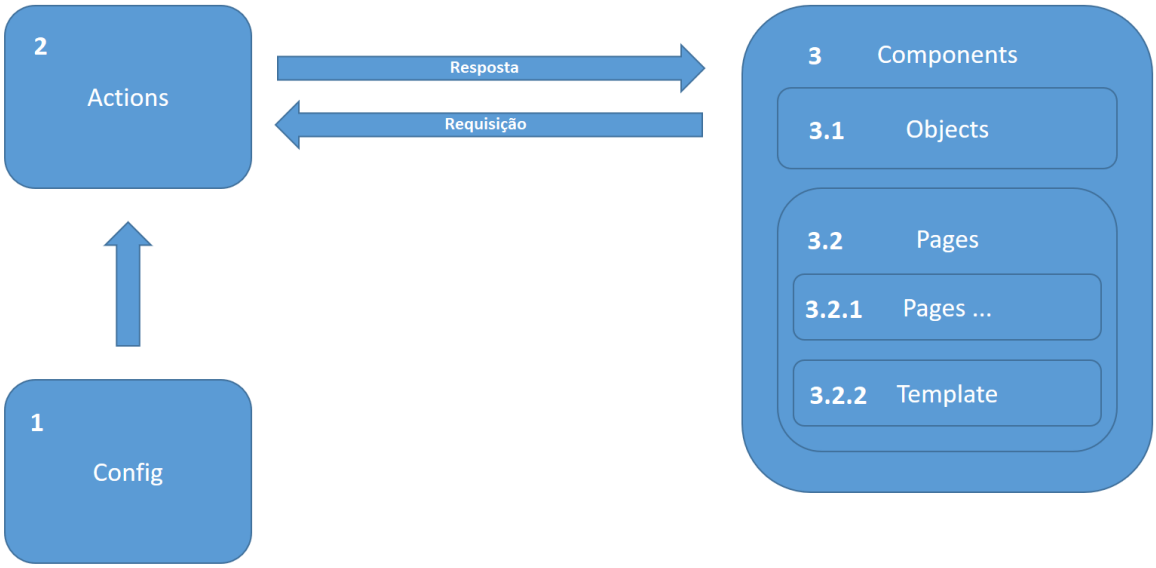


Fonte: Os Autores

O diagrama de implantação acima é dividido em duas partes. *Container* A representando os serviços utilizados no Firebase e o *Container* B onde consta a aplicação *React* desenvolvida em cima da arquitetura definida. Nota-se que a comunicação entre as tecnologias é feita através de conexão remota, que por sua vez tem as chamadas e configurações implementadas dentro da estrutura do sistema representado pelo *Container* B. O detalhamento do Firebase e seus serviços presentes no diagrama podem ser verificados no tópico 2.4.5 Firebase.

Para maior compreensão da forma que os dados são transmitidos e para facilitar a abstração dos componentes foi definida a arquitetura demonstrada na Figura 17, onde é demonstrado a divisão de responsabilidades contidas no sistema:

Figura 17 - Arquitetura do Sistema



Fonte: Os Autores

1. Config: O repositório config possui configurações de dependências ou bibliotecas externas. Inclusive a configuração para comunicação da aplicação com o Firebase ocorre neste módulo do sistema.

2. Actions: O repositório *actions* possui os arquivos de regras e requisições de dados para componente do sistema.

3. Components: Possui a estrutura organizacional dos componentes renderizados em tela.

Diferente de outras arquiteturas usualmente mais presentes em projetos web como MVC, por exemplo, as tecnologias escolhidas abstraem a necessidade do desenvolvimento do backend de forma separada. Deste modo toda a responsabilidade do servidor backend fica terceirizada para o Firebase.

3.1.8.3 Configuração do projeto para atender a arquitetura proposta

Após a arquitetura devidamente definida foi realizada a configuração do projeto para atender a arquitetura proposta.

A primeira etapa realizada foi a organização dos diretórios pelos quais estarão contidas as regras para funcionamento satisfatório da aplicação. A Figura 18, abaixo, demonstra os

diretórios gerados ao iniciar uma nova aplicação no *ReactJs* gerado através no *NPM*. A única exceção é que o arquivo App.js e Index.js deveriam estar no diretório raiz, porem foram colocados dentro da pasta src criado manualmente.

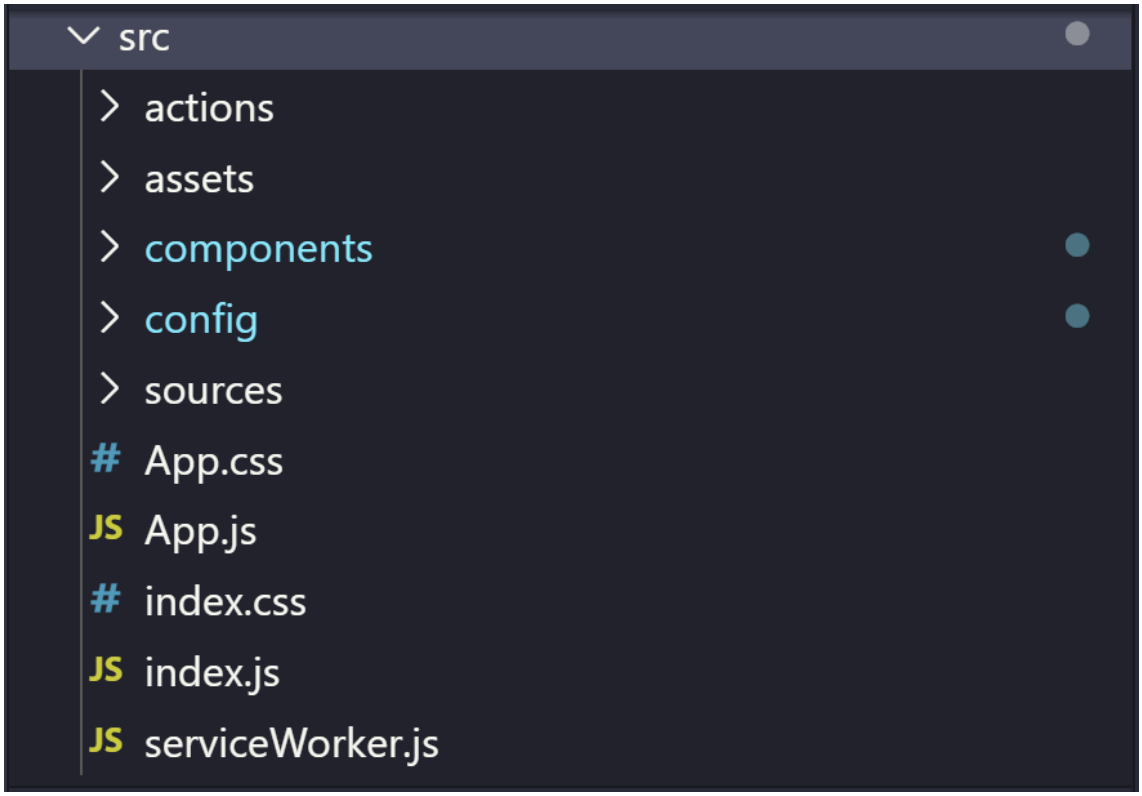
Figura 18 - Diretórios gerados ao iniciar uma nova aplicação ReactJs

Fonte: Os Autores

- Node_modules: possui todos os arquivos de dependências externas adicionadas ao projeto.
- Public: possui os arquivos públicos disponibilizados no cliente como index.htm *favicon* entre outros.
- Src: este diretório estão contidos os principais arquivos do projeto e é onde está definida toda a estrutura da arquitetura do sistema.
- Package.json/yarn.lock: Possui a configuração das dependências que rodam no projeto. É semelhante ao POM.XML para quem desenvolve em Java.

Um dos principais diretórios da aplicação é a pasta src (Figura 19) nela está contida os arquivos necessários para a execução do projeto.

Figura 19 - Diretório src



Fonte: Os Autores

Ao gerar um projeto em *React* são gerados alguns arquivos para a execução do projeto, são eles:

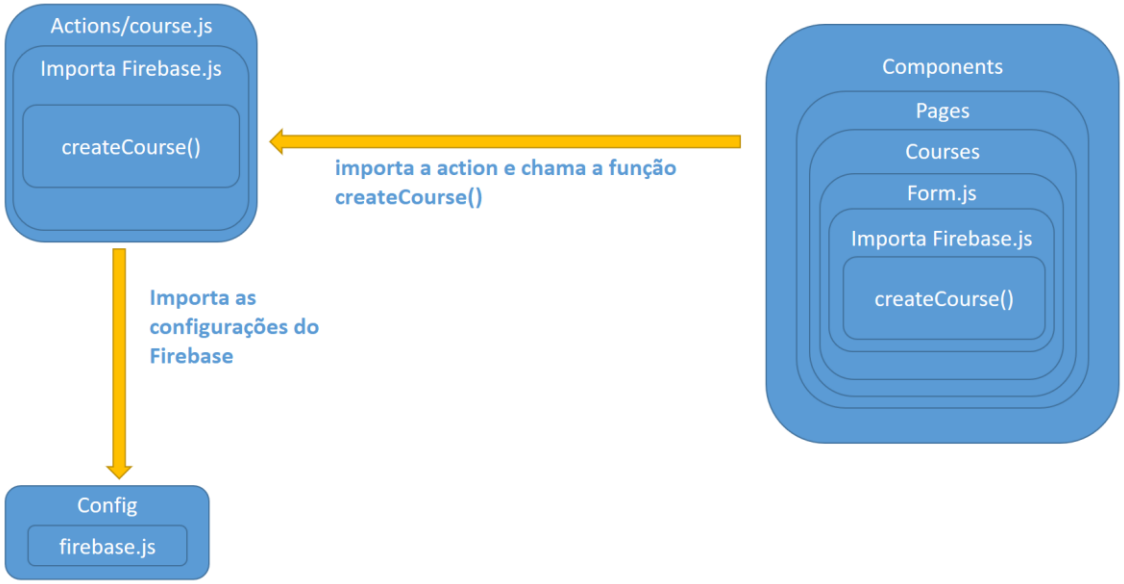
- App.js: Primeiro componente criado e instanciado no index.js;
- App.css: Contém os atributos de estilização do componente;
- Index.js: Renderiza o componente App dentro da div do index.html contido na pasta public;
- Index.css: Contém os atributos de estilização do componente;
- serviceWorker.js: Gerencia o *cache* da aplicação para funcionamento mais satisfatório. Quando configurado salva em cache os *assets* da aplicação para funcionamento mais performático.

Os demais diretórios foram definidos manualmente para atender a arquitetura definida para o projeto. São eles:

- *actions*: Estão contidas as regras de dados do sistema.
- *assets*: Dentro deste diretório estão contidos os arquivos de estilização e libs externas como o boostrap e fontes awesome pro exemplo.
- *components*: Possui os componentes desenvolvidos.
- *config*: Contém o arquivo de configuração do Firebase.
- *Sources*: Diretório onde estão localizados logos e imagens utilizados na aplicação.

O diagrama a seguir (Figura 20) demonstra o funcionamento prático da arquitetura para cadastro de um curso.

Figura 20 - Funcionamento da arquitetura para cadastro de curso

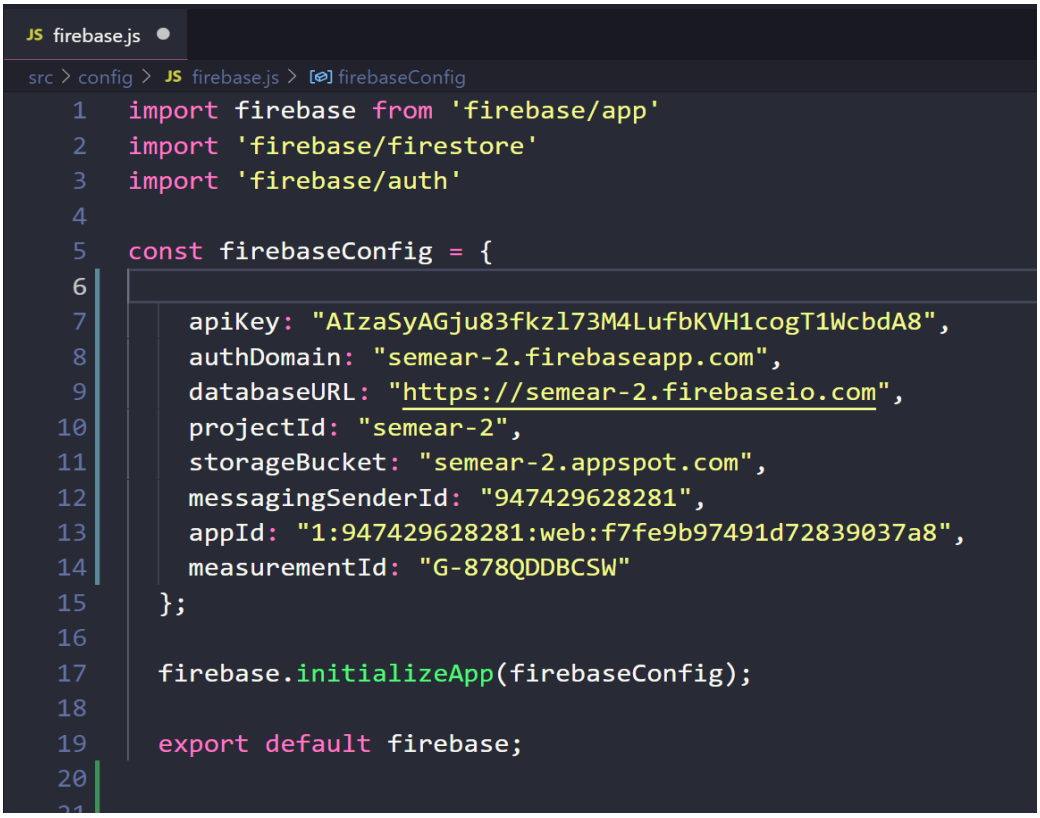


Fonte: Os Autores

Verificando a estrutura acima percebe-se que a requisição é unidirecional, ou seja, o evento parte do componente *Form* e chama em modo cascata os métodos necessários para executar a ação desejada.

Na prática a implementação fica da seguinte maneira: O arquivo *Config/Firebase.js*, representado na Figura 21, exporta a variável *firebase* que inicializa as configurações para conexão com o serviço.

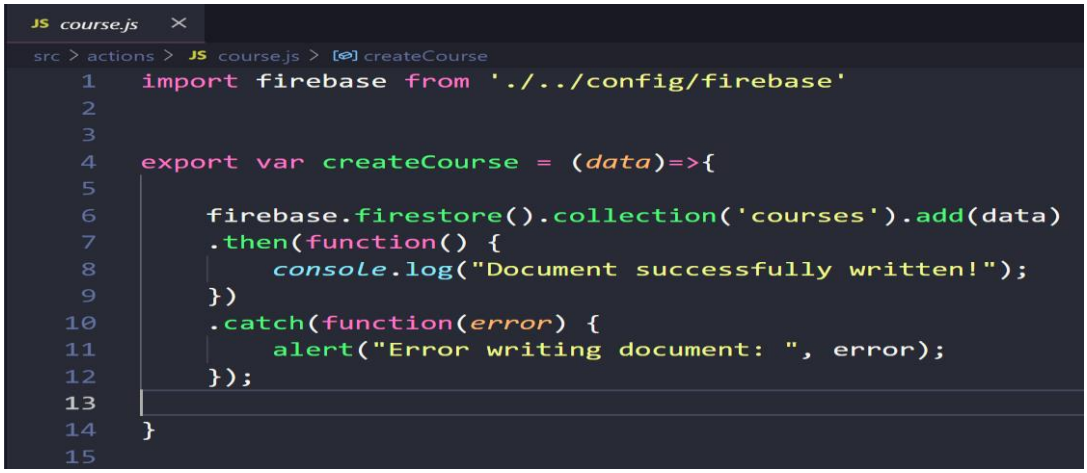
Figura 21 - Arquivo Config/Firebase.js



Fonte: Os Autores

Na linha 1 é realizado a importação da configuração do firebase e na linha 4 é definida a função createCourse() responsável por criar um novo curso.

Figura 22 - Arquivo Actions/Course.js:

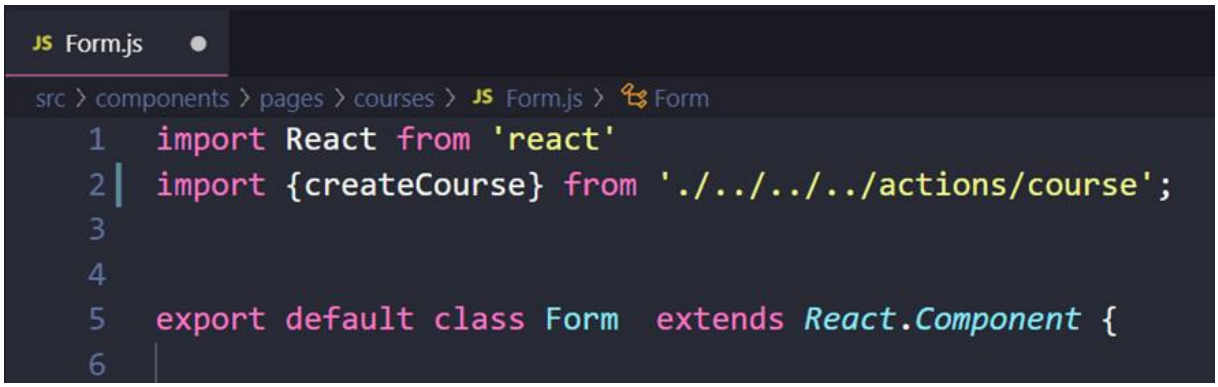


Fonte: Os Autores

As imagens abaixo, Figura 23 e Figura 24, representam o arquivo de construção do formulário de cadastro de Curso. Na linha 2 (Figura 23) a função createCourse() é importada

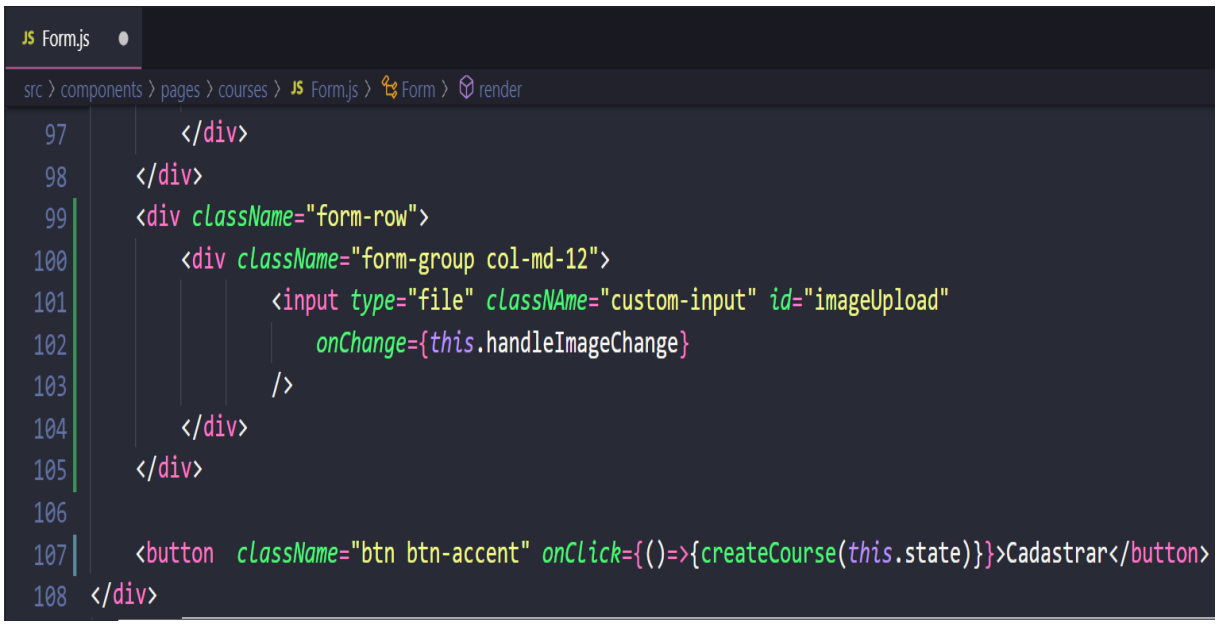
por desestruturação pronta para ser chamada. Na linha 107 (Figura 24) do componente a função createCourse() é chamada passando como parâmetro state do componente onde possui todas os dados informados no formulário.

Figura 23 - Importação da Função createCourse()



Fonte: Os Autores

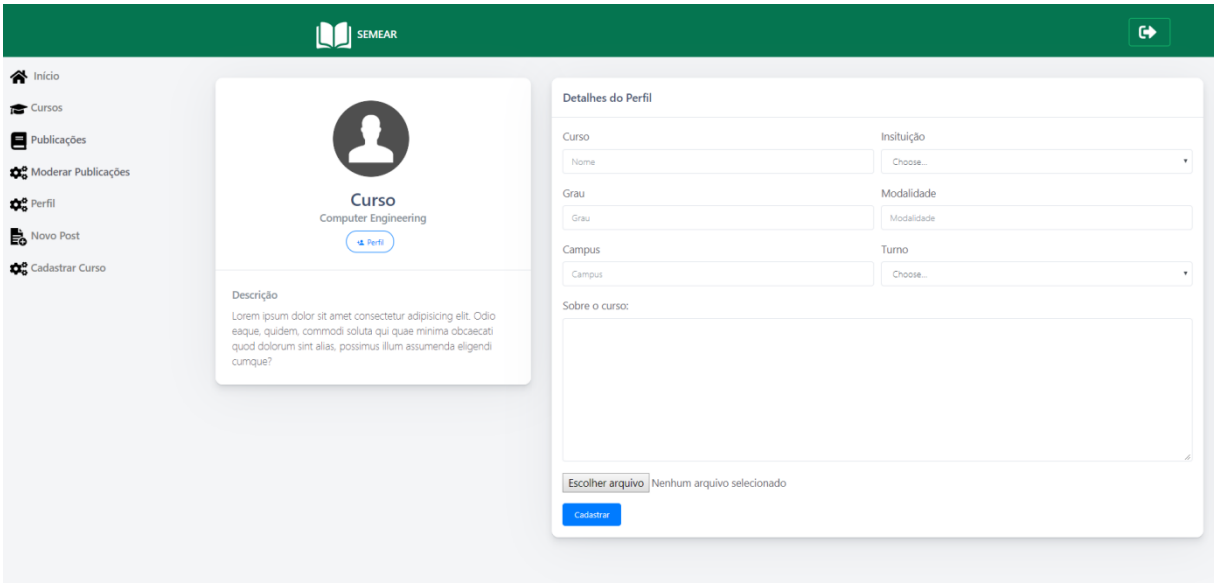
Figura 24 - Chamada da função createCourse



Fonte: Os Autores

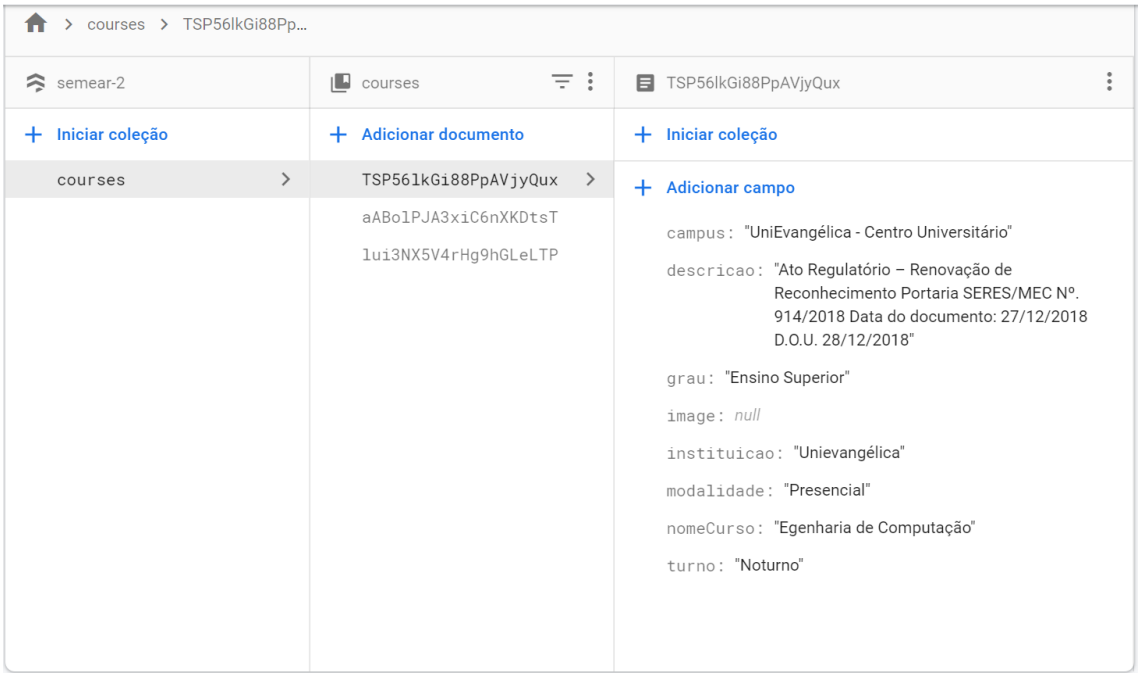
Ao final da execução do fluxo têm-se todas as informações inseridas no banco de dados do Firebase como representado na Figura 26. A Figura 25 apresenta a tela desenvolvida para o requisito UC016 – Cadastrar Curso, as demais telas do sistema estão disponíveis no APÊNDICE E.

Figura 25 - Cadastro Curso



Fonte: Os Autores

Figura 26 - Dados registrados no Firebase



Fonte: Os Autores

3.1.9 Sprint 2

No desenvolvimento da *sprint* 2, foi definido o requisito de autenticação de usuário. Este é o único requisito que não segue o padrão detalhado no requisito de modelo. Entretanto a única diferença notável é que ao invés de invocar os métodos de autenticação através *action*, o componente importa as configurações do *Firebase* diretamente ao seu escopo como pode

ser visto na Figura 27. Deste modo foram definidos os métodos de autenticação dentro do escopo do componente, sendo eles *logIn* (Figura 30) e *signOut* (Figura 31).

Figura 27 - Importação de recursos de autenticação

```
1  import React from 'react';
2  import {Link, Redirect}  from 'react-router-dom'
3  import firebase from './../../../config/firebase';
4
5
6  export  default class Header extends React.Component{
7
8      auth = {
9          user:"",
10         password:""
11     }
12
```

Fonte: Os Autores

Para o controle do ciclo de vida do componente, foi definido dentro do *constructor* o *state* que recebe dois atributos *logado* e *route*. O atributo *logado* recebe o valor *default false* e o atributo *route* recebe como valor inicial "/" especificando a rota raiz da aplicação. Veja na Figura 28 a implementação do método *constructor* do componente *header*.

Figura 28 - Método constructor do componente Header

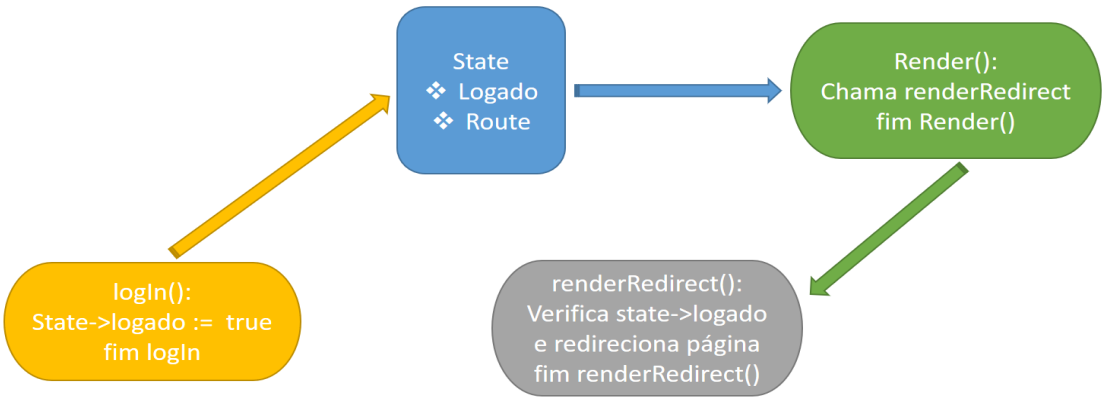
```
13      constructor(props){
14          super(props);
15          this.state = {
16              logado: false,
17              route: "/"
18          }
19      }
```

Fonte: Os Autores

Como mencionado no tópico 2.4.2 React o *state* é um dos atributos mais importantes de cada componente. É no *state* que é definido o ciclo de vida das informações carregadas no módulo desenvolvido. Toda vez que algum atributo do state é alterado, automaticamente o método responsável por renderizar os componentes no DOM é carregado fazendo que todas as informações sejam atualizadas em tela. Por este motivo o componente realiza a execução de métodos que implicam no comportamento da aplicação mesmo que não estejam diretamente relacionados.

Ao executar o método logIn() o estado do componente é alterado, ao ser alterado o método *render* é chamado novamente e executa o método renderRedirect() que é responsável por monitorar o valor do atributo logado do state. Caso seja *true* o método renderRedirect() retorna a rota definida e redireciona o usuário para outra tela. A Figura 29 ilustra como foi definido o ciclo de vida do componente *header*.

Figura 29 - Ciclo de vida do componente Header



Fonte: Os Autores

A Figura 30 abaixo descreve como foi feita a implementação do método de login. Repare que a função logIn verifica se os campos de user e password estão vazios, caso estejam preenchidos, é invocado o método auth()signInWithEmailAndPassword() que recebe como parâmetro o e-mail e a senha. Caso reja retornado verdadeiro o resultado é acessado através da promisse retorna uma response. Ao retornar, o estado é alterado onde o atributo logado recebe true.

Figura 30 – Implementação do método de logIn

```
21    logIn = () => {
22
23        if(this.auth.user !== '' && this.auth.password !== ''){
24            firebase.auth().signInWithEmailAndPassword(this.auth.user, this.auth.password)
25            .then((response) =>{
26
27                this.setState({logado: true});
28
29            }).catch(function(error) {
30                alert(error);
31            });
32        }
33    }
```

Fonte: Os Autores

Figura 31 – Implementação do método de signOut

```
41    signOut = () =>{
42        firebase.auth().signOut().then(function() {
43          deleteStorage()
44        },
45        function(error) {
46          console.error( error );
47        });
48    }
```

Fonte: Os Autores

Na Figura 32 – Médoto renderRedirect, pode ser viso o comportamento do método renderRedirect() quando invocado pelo componente. Note que se o atributo logado do state for igual a true, então o usuário é redirecionado para a página de áreas.

Figura 32 - Método renderRedirect

```
43
44      renderRedirect = () => {
45          if (this.state.logado === true) {
46            return <Redirect to='/areas' />
47          }
48      }
```

Fonte: Os Autores

3.1.10 Sprint 3

3.1.10.1 Desenvolvimento dos Casos de Uso UC002, UC009 e UC010

A Sprint 03 deu-se início na reunião de planejamento no qual foram selecionados os seguintes requisitos para o desenvolvimento: UC002 – Manter Cadastro, UC009 - Manter Trabalhos e UC010 – Inserir Descrição do Artigo. Os mesmos foram antepostos de acordo com as prioridades de funcionalidades para a prova de conceito.

Estes requisitos foram definidos como prioridade essencial e têm como base de desenvolvimento o modelo apresentado na seção 3.1.8.2 Requisito modelo e Arquitetura do sistema. Após a definição das prioridades foi anexado no Kanban quais atividades necessárias para desenvolvimento de cada caso de uso.

4 RESULTADOS ALCANÇADOS

Durante o desenvolvimento deste trabalho foram planejadas e executadas diversas atividades, nas quais foram produzidos artefatos que compõem o resultado deste trabalho, no entanto, nem todos os itens do *backlog* do produto foram construídos. A Figura 33 possui uma representação gráfica de todas as atividades planejadas destacando o que já foi feito e que será realizado posteriormente a entrega deste documento.

Figura 33 – Backlog do Produto

Fonte: Os Autores

Como complemento do exposto, segue a lista de casos de uso não implementados e seus respectivos impedimentos:

- UC 001 Manter Moderadores: Tem como pré-requisito o cadastro de usuários, publicações para implementação.
- UC 002 Manter Cadastro:
- UC 003 Analisar publicações do artefato: Tem como pré-requisito o UC 001.
- UC 004 Aceitar publicações do artefato: Tem como pré-requisito o UC 001
- UC 005 Rejeitar publicação: Tem como pré-requisito o UC 001

- UC 006 Envio de e-mails informando a situação: Estava fora das *sprints* iniciais do projeto.
- UC 010 Inserir descrição do que será tratado no artigo: Dificuldades na implementação com novas tecnologias.
- UC 012 Avaliar trabalhos: Tem como pré-requisito o UC 004
- UC 013 Inserir comentários. Tem como pré-requisito o UC 010.

5 CONSIDERAÇÔES FINAIS

A falta de um meio eficiente de divulgação dos trabalhos produzidos no meio acadêmico causam diversos níveis de insatisfação que vão desde os leitores, autores, até as bibliotecas, nesse sentido a construção de meio eficiente, através do uso de tecnologia, se faz necessário a fim de proporcionar melhores resultados a todos os envolvidos.

A realização desse projeto viabilizou esta ideia e sustentou um grande crescimento pessoal dos autores oportunizando a aplicação prática das teorias adquiridas no decorrer da graduação, através de pesquisas e utilização de ferramentas e métodos para resolução do problema proposto. No entanto, devido ao planejamento de algumas estimativas imprecisas e impedimentos no decorrer do desenvolvimento do projeto, algumas funcionalidades importantes para completar a prova de conceito não foram implementadas durante o desenvolvimento deste trabalho.

Deste modo a implementação dos requisitos faltantes se dará pela continuidade do desenvolvimento do projeto pelos autores, ou terceiros, seguindo os conceitos abordados neste trabalho, a fim de obter-se a conclusão do mesmo e por fim a disponibilização da prova do mesmo.

Foram encontradas diversas dificuldades durante todo o processo de desenvolvimento deste trabalho, mas foi de grande valia para formação profissional a oportunidade de colocar em prática os conceitos aprendidos durante a graduação. A falta de experiência e maturidade nas foram fatores que impactaram todo o processo de desenvolvimento do trabalho. Outra dificuldade encontrada foi a curva de aprendizado de uma nova tecnologia com paradigmas de desenvolvimento próprios; a adaptação na implementação de uma tipologia de consumo de serviço mais recente disponível no mercado.

Fazendo uma analise do que foi aprendido durante o desenvolvimento deste trabalho, é possível destacar que escolhas tomadas inicialmente têm influência durante toda a vida útil do projeto e até mesmo da aplicação. No entanto, apesar de todas as dificuldades encontradas durante a elaboração deste trabalho destaca-se o aprendizado como o principal ganho, sendo este de valor inestimável na formação dos autores.

REFERÊNCIAS BIBLIOGRÁFICAS

BECK, K. et al. **Manifesto para o desenvolvimento ágil de software.** 2001. Disponível em: <http://www.manifestoagil.com.br/> Acesso em: 11 mai. 2019.

CÂMARA, M. et al. **O que você deve saber sobre o funcioanmento do React Native.** 2018. Disponível em: < https://tableless.com.br/o-que-voce-deve-saber-sobre-funcionamento-react-native//> Acesso em: 12 mai. 2019.

COHN, M. User Stories Applied For Agile Software Development. [S.l.]: Addison-Wesley Professional, 2004.

ECMA-INTERNATIONAL. ECMA-262. 2018, 9 ed. Disponível em: <https://www.ecma-international.org/ecma-262/9.0/index.html>. Acesso em: 17 mai. 2019

EGESTOR.O que é e como funciona o método. 2017. Disponível em: <https://blog.egestor.com.br/o-que-e-e-como-funciona-o-metodo-kanban/> Acesso em: 12 junh. 2019.

FILHO, Agenor. et al. **Importância dos repositórios institucionais na preservação intelectual: em foco a gestão do conhecimento**. In: XXXV ENCONTRO NACIONAL DE ESTUDANTES DE BIBLIOTECONOMIA, DOCUMENTAÇÃO, CIÊNCIA DA INFORMAÇÃO E GESTÃO DA INFORMAÇÃO ESCOLA DE CIÊNCIA DA INFORMAÇÃO – UNIVERSIDADE FEDERAL DE MINAS GERAIS. Belo Horizonte: UFMG, 2012. Disponível em: <http://www.brapci.inf.br/index.php/res/download/98803> Acesso em: 03 nov. 2019

FIREBASE. Adicionar o Firebase ao projeto em JavaScript. Disponível em: <https://firebase.google.com/docs/web/setup?hl=pt-BR>. Acesso em: 17 mai. 2019.

GIT. **Pro Git**. Disponível em: <https://git-scm.com/book/pt-br/v2>. Acesso em: 13 mai. 2019.

GOOGLE. **Firebase helps mobile and web app teams succeed.** Disponível em: <https://firebase.google.com/?gclid=Cj0KCQjwt_nmBRD0ARIsAJYs6o3nmxu8fc7tZjxQB09H7mMsee9KWB0EKD-iCVwGUdk31UlHANlBvSYaAgcIEALw_wcB>. Acesso em 15 mai. 2019.

GRADUS. 7 Razões irresistíveis para implantar rápido. 2018. Disponível em: <https://www.gradusct.com.br/kanban/> Acesso em: 12 junh. 2019.

HANASHIRO, Akira. **O que se pode fazer com JavaScript hoje em dia?.** 2018. Disponível em: <https://www.treinaweb.com.br/blog/o-que-se-pode-fazer-com-javascript-hoje-em-dia/>. Acesso em: 18 mai. 2019.

HELM, R.; WILDT, D. **Históricas de Usuário:** Por que e como escrever requisitos de forma ágil? 3. Ed. Disponível em: <https://gallery.mailchimp.com/6f3ad17c3448b982a9c49fa6e/files/historias_de_usuario_3_0.pdf>. Acesso em 12 mai. 2019.

HUMBLE, J.; FARLEY, D. **Entrega contínua:** como entregar software de forma rápida e confiável. 8. Ed. Porto Alegre: Bookman, 2014.

Instrumento de Avaliação Institucional Externa: Presencial e a Distância. Brasília: Inep/MEC, 2017.

LIBÂNEO, J. C. **A avaliação escolar.** São Paulo: Cortez, 1994.

LIBÂNEO, J. C. **O processo de ensino na escola**. São Paulo: Cortez, 1994.

LIBÂNEO, J. C. **Os métodos de ensino.** São Paulo: Cortez, 1994.

LIMA, M. **O Guia Completo do React e o seu Ecossistema.** 2017. Disponível em: <https://tableless.com.br/guia-completo-react-ecossistema/>. Acesso em: 15 mai. 2019.

NEVES, J. L. **Pesquisa Qualitativa**: Características, Usos e Possibilidades. Caderno de Pesquisas em Administração, v. 1, n. 3, 1996.

PRESSMAN, R.; MAXIM, B. **Engenharia de Software:** Uma Abordagem Profissional. 8. Ed. Porto Alegre: McGraw-Hill, 2016.

PRIKLADNICKI, R.; WILLI, R.; MILANI, F. **Métodos Ágeis para Desenvolvimento de Software.** 3. ed. Porto Alegre: Bookman, 2014.

PROSSER, M.; TRIGWELL, K. **Development and Use of the Approaches to Teaching Inventory. Educational Psychology**, 2004. Disponível em < https://doi.org/10.1007/s10648-004-0007-9> Acesso em: 10 jun. 2019.

REACT, **Crie um novo React App**. Disponível em:<https://pt-br.reactjs.org/docs/create-a-new-react-app.html>. Acesso em: 22 out. 2019.

SADALAGE, Pramond J; FOWLER, Martin. **NoSQL Essencial:** Um Guia Conciso para o Mundo Emergente da Persistência Poliglota. Primeira. Ed. São Paulo, SP – Brasil: Novatec, 2013. 2016 p. v. 1.

SCHWABER, K; SUTHERLAND, J. **Guia do Scrum.** 2017. Disponível em: < https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Portuguese-Brazilian.pdf > Acesso em: 11 mai. 2019.

SILVEIRA P,GUILHERME S, LOPES S, STEPPAT N, Kung F. **Introdução a Arquitetura e Design de Software – Uma visão sobre a plataforma Java**, v. 1, n. 3, 2011.

SMART, J. **BDD in Action:** Behavior-Driven Development for the Whole Software Lifecycle. Shelter Island, NY: Manning Publications, 2014.

SOLÍS, C.; WANG, X. **A Study of the Characteristics of Behaviour Driven Development.** 2011. Disponível em: <https://ulir.ul.ie/bitstream/handle/10344/1256/Solis_2011_behaviour.pdf>. Acesso em: 12 mai. 2019.

SOMMERVILLE, I. **Engenharia de Software.** 9. ed. São Paulo: Pearson Prentice Hall, 2011.

STEPPAT, Nico; **Banco de Dados não relacionais e o movimento NoSQL.** Disponível em: <http://blog.caelum.com.br/bancos-de-dados-nao-relacionais-e-o-movimento-nosql/>. Acesso em: 30 abr. 2018.

VIEIRA, M. et al. **Bancos de Dados NoSQL:** Conceitos, Ferramentas, Linguagens e Estudos de Casos no Contexto de Big Data. Disponível em: <http://data.ime.usp.br/sbbd2012/artigos/pdfs/sbbd_min_01.pdf/>. Acesso em: 30 abr. 2018.

WODEHOUSE, Carey. **SQL vs. NoSQL Databases:** What's the Difference? Disponível em: <https://www.upwork.com/hiring/data/sql-vs-nosql-databases-whats-the-difference/>. Acesso em: 20 mai. 2019.

APÊNDICES

APÊNDICE A – DOCUMENTO DE VISÃO

DOCUMENTO DE VISÃO

SEMEAR – Sistema para publicação e compartilhamento de trabalhos acadêmicos

VERSÃO: 1.0

Autores:

Davi Gonçalves de Souza
Marcos Moraes Pereira

Anápolis – GO

2019

Sumário

1. Objetivo *53*

2. Stakeholders *53*

3. Descrição do produto *53*

4. Escopo do produto *54*

5. Fora do escopo do produto *54*

6. Envolvimento *54*

6.1. Abrangência 54

6.2. Papel dos atores 54

6.3. Analista 55

7. Requisitos Funcionais *56*

8. Requisitos Não Funcionais *56*

9. Proposta de Solução Tecnológica Escolhida *57*

10. Custo da Proposta *57*

11. Cronograma de execução *58*

12. Restrições *58*

13. Termo de aceitação *58*

14. Histórico de Versões *58*

1. Objetivo

O propósito deste documento é coletar, analisar e definir as necessidades e características do sistema SEMEAR, focando nas potencialidades requeridas pelos desenvolvedores e usuários-alvo, e como estes requisitos foram abordados no sistema. A visão do sistema documenta o ambiente geral de processos desenvolvidos para o sistema, fornecendo a todos os envolvidos uma descrição compreensível deste e suas macro-funcionalidades.

2. Stakeholders

Maio de 2019		
Nome	**Função**	**Responsabilidade**
Marcos Moraes Pereira	Desenvolvedor e Scrum Master	Responsável pela codificação e execução do Scrum
Davi Souza	Product Owner	Responsável pelo Product Backlog
Millys Fabrielle Araújo Carvalhaes	Cliente	n/a

3. Descrição do produto

O SEMEAR é uma aplicação que permite inicialmente armazenar e disponibilizar os diversos documentos produzidos pelos alunos do curso de Engenharia de Computação/Software, tornando-os uteis para o aprendizado de outrem.

Com este propósito o SEMEAR não será apenas um repositório, mas sim uma rede que possibilitará a interação entre usuários através da publicação e compartilhamento de trabalhos, a fim de auxiliar e estimular o aprendizado e a geração de novos conhecimentos.

Possuirá um repositório de artigos, documentos desenvolvidos ou não em sala de aula pelos discentes do curso de Engenharia de Computação.

Também oferece um histórico de contribuições realizadas por cada usuário cadastrado no sistema. Onde essas informações serão utilizadas para gerar indicadores sobre os membros da plataforma.

Disponibiliza cadastro de alunos e professores aplicando controle de funcionalidades onde serão aplicadas restrições de acordo com cada perfil cadastrado.

Possibilita a comunicação com outros sistemas baseados em serviços e ou microserviços que ofereçam APIs baseadas no protocolo HTTP.

4. Escopo do produto

O sistema deverá ser capaz de fornecer os seguintes serviços para atender as necessidades dos interessados:

- O sistema deve permitir upload de arquivos;
- O sistema deve ser capaz de cadastrar usuários;
- O sistema deve ser capaz de realizar download dos arquivos publicados.
- O sistema deverá se comunicar com sistemas de terceiros utilizando APIs REST;
- O sistema deverá ser capaz de guardar log das atividades dos usuários cadastrados na base de ados;
- O sistema deverá oferecer as mesmas funcionalidades independente da plataforma;
- O sistema deverá possibilitar iterações entre os integrantes por texto no formato de comentário ou por avaliações.

5. Fora do escopo do produto

O sistema não deverá fornecer os serviços para atender as necessidades dos interessados:

- O sistema não dará suporte ao upload de videos.
- O sistema não dará suporte ao upload de arquivos com formatos diferente das extensões definidas nos requistos.

6. Envolvimento

6.1. Abrangência

Inicialmente o SEMEAR será disponibilizado como sistema piloto para o curso de Engenharia de Computação, porém com possibilidade de expansão para os demais cursos da instiuição após a comprovação da viabilidade e aceitação em relação aos acadêmicos que utilizarão a plataforma.

6.2. Papel dos atores

Atores são pessoas, equipamentos ou outros sistemas que interagem como sistema em questão, enviando ou recebendo mensagens. Abaixo estão descritos de forma resumida o papel dos atores do sistema SEMEAR.

- **Usuário**

Descrição	**O usuário é o ator que irá utilizar o sistema**
Papel	O sistema SEMEAR será mais eficiente se seu repositório estiver constantemente utilizado a fim de manter uma quantidade significativa de artefatos em seu acervo. O usuário aluno é de suma importância para a manutenção do conteúdo disponível no sistema. O usuário professor é de suma importância para a manutenção e avaliação dos trabalhos publicados no sistema. O usuário administrador é de suma importância para a operação de funcionalidades de cadastros e de conteúdo no sistema.
Insumos ao Sistema	• Trabalhos desenvolvidos pelos alunos da instituição • Indicadores de efetividade na participação de cada usuário cadastrado no sistema • Avaliação dos trabalhos publicados • Comentários salvos pelos usuários • Dados de cadastro dos usuários • Informações sobre o funcionamento do sistema (usabilidade)
Representante	Definido pelo departamento responsável pela infra estrutura ou gerencia do sistema.

6.3. Analista

Descrição	**Fornece as especificações de funcionalidades que o sistema deve atender.**
Papel	Descrever de forma detalhada todos os requisitos do sistema.

Insumos ao Sistema	• Informações sobre regras de negócio; • Requisitos do sistema; • Aprovação do projeto e implementação.
Representante	Marcos Moraes Pereira

7. Requisitos Funcionais - RF

Nesta seção são descritas as principais necessidades e funcionalidades requeridas pelas partes interessadas.

Nº	Nome	Descrição
RF001	Manter Usuário	Permite cadastrar, alterar, visualizar e excluir as informações a respeito de usuários do sistema SEMEAR que serão ou já foram cadastrados no sistema.
RF002	Manter Artefatos	Permite cadastrar, alterar, visualizar e excluir os arquivos anexados pelos usuários que possuem o perfil para tal função dentro do sistema.
RF003	Autenticação	Permitir que o usuário realize autenticação no sistema, para executar funcionalidades de acordo com as restrições definidas no sistema.
RF004	Notificação	Permitir que usuários autenticados no sistema sejam notificados a respeito de publicações que foram curtidas ou marcadas como favoritas.

8. Requisitos Não Funcionais RFN

Nº	Nome	Descrição
RNF001	Usabilidade	O sistema deve facilitar para o usuário a compreensão da funcionalidade disponível através do uso de termos comuns dentro do meio no qual ele está inserido. Deverá possuir uma apresentação simples tornando a realização das ações de forma intuitiva e lógica. Deve possuir ajuda online e documento de fácil acesso pelo usuário.
RNF002	Desempenho	

		O sistema deverá executar de forma satisfatória levando até 10 segundos no carregamento de conteúdos na interface.
RNF003	Confiabilidade	Todas as informações sobre valores e, resultados das execuções de cada ação do sistema que gerem valores devem ser precisos. O sistema deve possuir sistema de recuperação de falhas
RNF004	Portabilidade	O sistema deverá ser executado nas plataformas web e mobile sem perda nenhuma de eficiência em relação as funcionalidades.

9. Proposta de Solução Tecnológica Escolhida

O sistema será desenvolvido utilizando como biblioteca de interfaces o ReactJS para o módulo web e ReactNative para o desenvolvimento do módulo mobile compativeis com Android e IOS. Para o armazenamento de dados será utilizado o banco Firebase Firestore da Google descartando a necessidade de outro servidor de aplicação.

Será necessário conexão com a internet para os módulos web e mobile.

No desenvolvimento serão utlizadas diversas ferramentas, entre elas: Visual Studio Code, Sublime, Notepad++, Astah, Android Studio, Genimotion. Word, WPS.

10. Custo da Proposta

A proposta de desenvolvimento do sistema inclui os seguintes itens:

- Análise de Requisitos;
- Projeto do sistema;
- Codificação do sistema;
- Testes e Validação;
- Documentação do Sistema;
- Implatação do Sistema;
- Treinamento Presencia;
- Correções e atualizações do sistema (por 7 dias após a entrega do treinamento).

O valor desta proposta não leva em consideração valores monetários.

11. Cronograma de execução

O cronograma abaixo exibe uma estimativa do desenvolvimento do sistema. Considerando a proposta aceita.

Atividade/ data	Mai	Jun	Jul	Ago	Set	Out	Nov
Análise de Requisitos	X						
Projeto do Sistema	X						
Codificação		X	X	X	X	X	
Testes e validação		X	X	X	X	X	
Documentação		X	X	X	X	X	
Implantação						X	
Treinamento						X	
Correções/Atualizações						X	X
Suporte ao Usuário							X

A data prevista para entrega do sistema, implantação e treinamento é :**10 a 15 de novembro de 2019.**

No ato da implantação, nem todas as funcionalidades poderão estar disponíveis ou completas, porém estas serão entregues dentro do prazo estipulado para correções e atualizações.

12. Restrições

Descreve as restrições que sejam impostas ao sistema ou ao processo de desenvolvimento.

- Os softwares utilizados para o desenvolvimento da aplicação deverão ser gratuitos.
- Para o desenvolvimento e homologação deverá ser utilizados serviços gratuitos ou com quotas de banda necessárias para dispensar custos com o desenvolvimento.

13. Termo de aceitação

Eu _____, certifico que estou ciente e de acordo com a proposta acima apresentada, tanto no que se refere as funcionalidades apresentadas no item 4 deste documento, quanto ao valor da proposta, presente no item 7, e com cronograma de execução do item 8.

14. Histórico de Versões

Data	Versão	Descrição	Autor	Aprovado Por
11/05/2019	1.0	Versão Inicial	Marcos	Davi
17/09/2019	2.0	Revisão do escopo	Davi	Marcos
04/11/2019	3.0	Correções pontuais de ortografia e revisão dos RNF	Davi	Marcos

APÊNDICE B – ESPECIFICAÇÃO DE CASOS DE USO

Caso de uso: Fazer upload de trabalhos
Descrição Geral: O caso de uso se inicia, quando o usuário aluno deseja efetuar um upload de algum artefato acadêmico para o sistema semear.
Atores: Aluno
Pré-condições: Aluno estar logado no sistema
Pós-condições (Garantia de sucesso): Upload concluído, descrição do artefato feito e aluno aguardando análise do artefato.
Fluxo Principal: 1. Aluno deseja efetuar um upload de algum artefato acadêmico; 2. O sistema solicita que o usuário Aluno insira uma descrição do que será tratado no artigo, juntamente com o anexo do artigo; 3. O aluno insere todas as informações solicitadas; 4. O sistema valida todos os campos; 5. O sistema cadastra os dados inseridos; 6. O sistema exibe uma mensagem de sucesso.
Fluxo Alternativo: 1. Dados incorretos 1.1. O sistema encontra algum dado incorreto e redireciona o usuário para o passo 2;
Fluxo de exceção: 1. Falha na conexão com o banco de dados 1.1. O sistema exibe a mensagem [“Falha ao conectar com o banco de dados”]; 1.2. O sistema redireciona o usuário ao passo 2 do fluxo principal.

Caso de uso: Visualizar Trabalhos
Descrição Geral: O caso de uso se inicia, quando o usuário aluno deseja visualizar algum artefato acadêmico do sistema;
Atores: Aluno
Pré-condições: Aluno estar logado no sistema, ter artefatos acadêmicos já cadastrados no sistema, artefatos terem sidos aprovados pelo moderador.
Pós-condições (Garantia de sucesso): Visualização concluída
Fluxo Principal: 1. Aluno deseja visualizar algum artefato acadêmico já cadastrado no sistema; 2. O sistema exibe o campo de pesquisa para o usuário inserir algum parâmetro de pesquisa sobre o artefato de sua escolha; 3. O aluno insere o parâmetro de sua escolha e inicia a pesquisa; 4. O sistema exibe todos os artefatos com base no parâmetro digitado pelo usuário; 5. O usuário seleciona o artefato de sua escolha; 6. O sistema exibe o artefato selecionado pelo usuário; 7. O usuário pode opcionalmente, avaliar o trabalho que está visualizando. 8. O usuário pode opcionalmente, inserir comentários no trabalho que está visualizando;

9. O usuário pode opcionalmente, fazer download do artefato selecionado.
Fluxo Alternativo: 1. Dados incorretos 1.1. O usuário não insira nenhum parâmetro para pesquisa, ou algum parâmetro desconhecido para o sistema, o sistema exibe uma mensagem de advertência e o usuário retorna ao passo 2 do fluxo principal.
Fluxo de Exceção: 1. Falha na visualização do artefato 1.1 O sistema exibe a mensagem [“Falha ao visualizar”]; o sistema redireciona o usuário ao passo 4 do fluxo principal.

Caso de uso: Analisar publicação do artefato
Descrição Geral: O caso de uso se inicia quando há artefatos publicados no sistema que necessitam de analise para possível aprovação.
Atores: Moderador, Servidor de e-mail.
Pré-condições: Moderador estar logado no sistema, possuir artefatos publicados que necessitam de análise.
Pós-condições: Artefato analisado, aluno ter recebido um e-mail informando a situação.
Fluxo Principal: 1. O moderador loga no sistema; 2. O sistema exibe todos os artefatos que estão aguardando analise; 3. O moderador seleciona o artefato que deseja analisar; 4. O moderador analisa o artefato e o aprova; 5. O servidor de e-mail envia um e-mail para o usuário informando-o, que seu artefato foi aprovado; 6. O sistema então disponibiliza o artefato para visualização;
Fluxo Alternativo: 1. Artefato rejeitado 1.1. Caso o moderador rejeite o artefato no passo 4 do fluxo principal, o moderador deverá esclarecer o motivo da rejeição e o servidor de e-mail irá enviar um e-mail informando o usuário que o publicou;

Caso de uso: Manter moderadores
Descrição Geral: O caso de uso se inicia quando o usuário administrador deseja manter o usuário moderador;
Atores: Administrador
Pré-condições: O administrador estar logado no sistema
Pós-condições: Nenhuma
Fluxo principal: 1. O usuário administrador deseja cadastrar um moderador no sistema; 2. O sistema exibe o formulário de cadastro; 3. O administrador preenche as informações;

4. O sistema valida as informações;
5. O sistema exibe mensagem de sucesso;

Fluxo Alternativo:

1. Dados incorretos;
   1.1. O sistema encontra algum dado incorreto e redireciona o administrador ao passo 2;
2. Excluir moderador;
   2.1. O administrador deseja excluir algum moderador do sistema;
   2.2. O administrador solicita ao sistema todos os moderadores ativos;
   2.3. O sistema exibe todos os moderadores;
   2.4. O administrador seleciona o moderador que ele deseja excluir;
   2.5. O sistema exibe as informações do moderador;
   2.6. O administrador exclui o moderador.
   2.7. O sistema armazena as novas informações
3. Alterar moderador
   3.1. O administrador deseja alterar alguma informação de um moderador do sistema;
   3.2. O administrador solicita ao sistema todos os moderadores ativos;
   3.3. O sistema exibe todos os moderadores;
   3.4. O administrador seleciona o moderador que deseja alterar;
   3.5. O sistema exibe as informações do moderador;
   3.6. O administrador faz alguma modificação;
   3.7. O sistema armazenas as novas informações;

Fluxo de exceção:

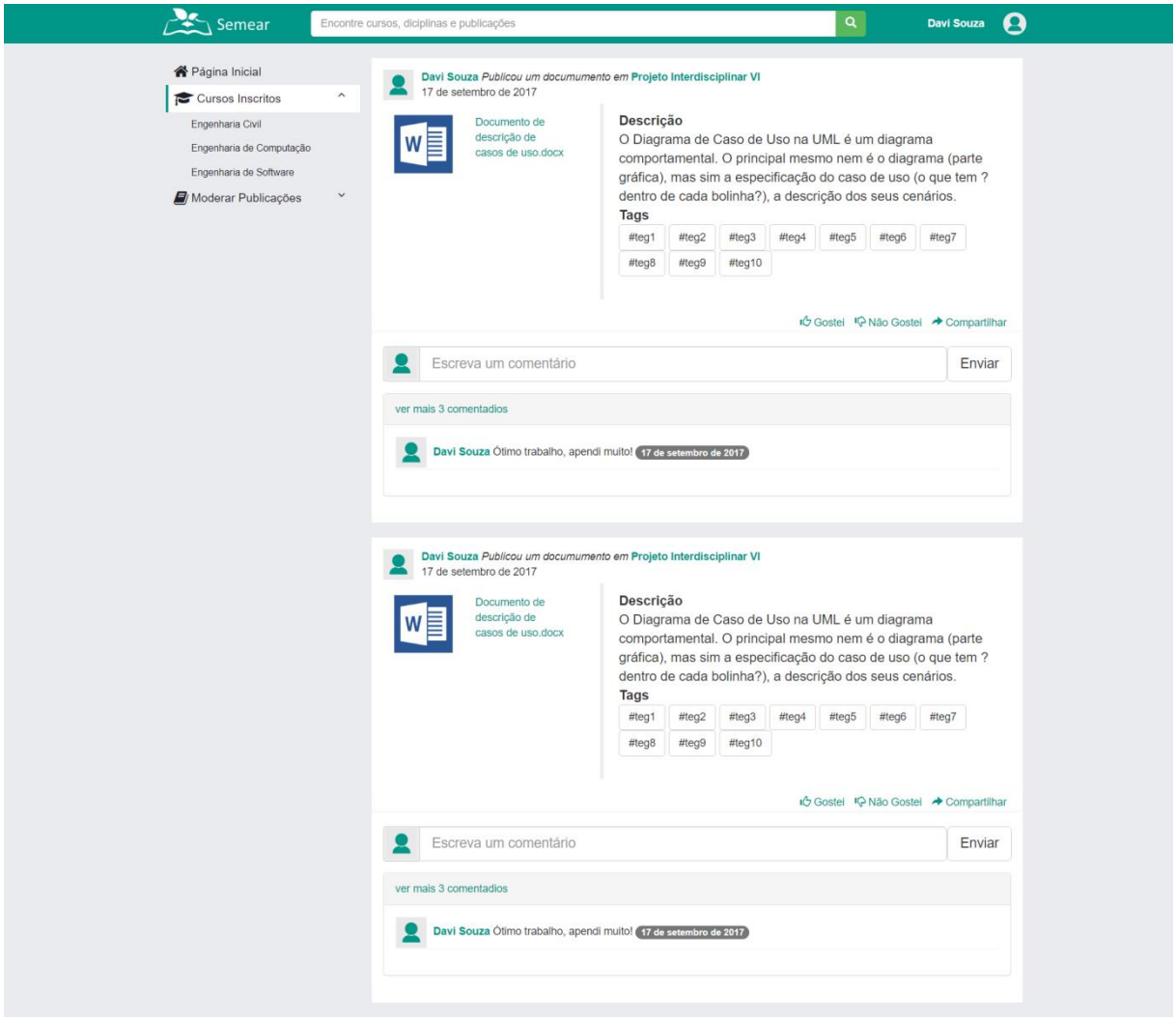
1. Falha na conexão com o banco de dados
   1.1. O sistema exibe a mensagem [“Falha ao conectar com o banco de dados”];

Caso de uso: Manter aluno
Descrição Geral: O caso de uso se inicia quando o moderador deseja cadastrar/excluir ou alterar um aluno.
Atores: Moderador, Servidor de e-mail.
Pré-condições: O moderador estar logado no sistema, possuir pré-cadastros para aprovação.
Pós-condições: Pré-cadastro aprovado
Fluxo principal: 1. O usuário moderador deseja aprovar um pré-cadastro no sistema; 2. O sistema exibe os pré-cadastros pendentes; 3. O moderador seleciona um pré-cadastro; 4. O moderador aprova o pré-cadastro; 5. O servidor de e-mail envia um e-mail para o usuário informando a situação;
Fluxo Alternativo: 1. Pré-cadastro reprovado; 1.1. O Servidor de e-mail envia um e-mail para o usuário informando a situação;
Fluxo de exceção: **2.** Falha na conexão com o banco de dados **2.1.** O sistema exibe a mensagem [“Falha ao conectar com o banco de dados”];

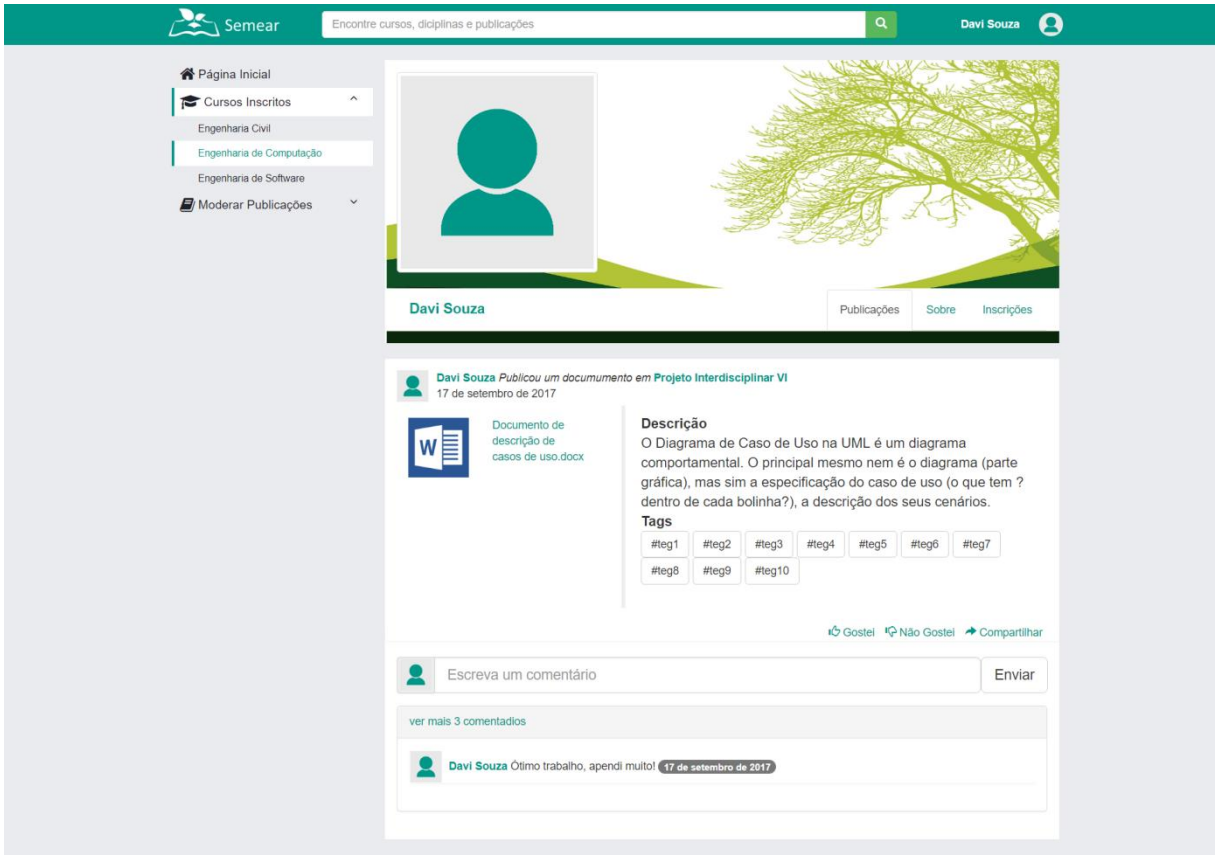
APÊNDICE C – PROTÓTIPOS

Login

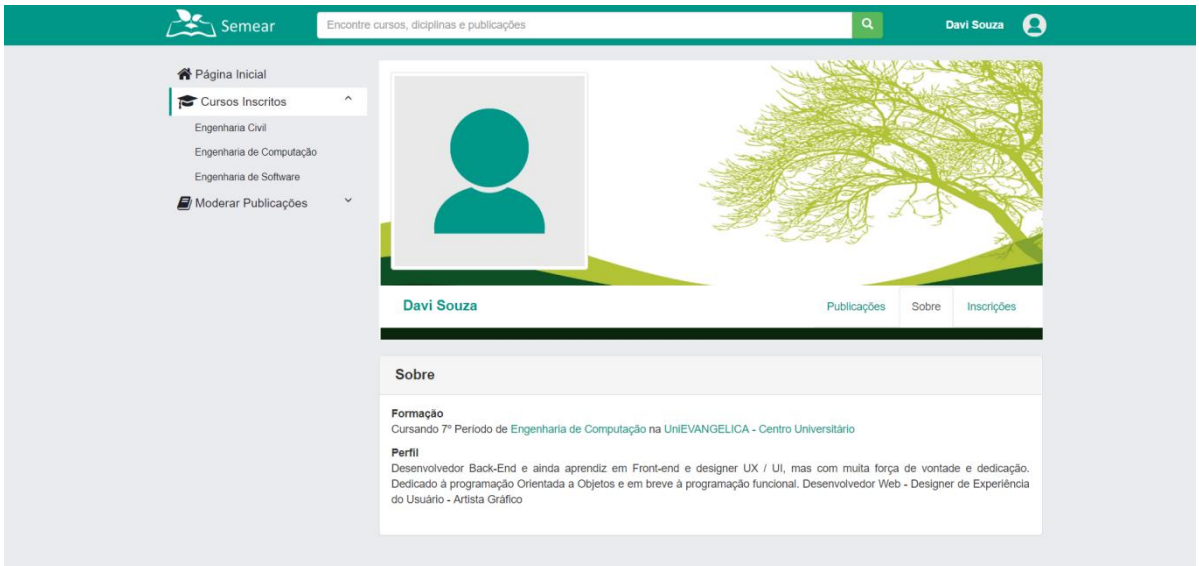
Página Inicial



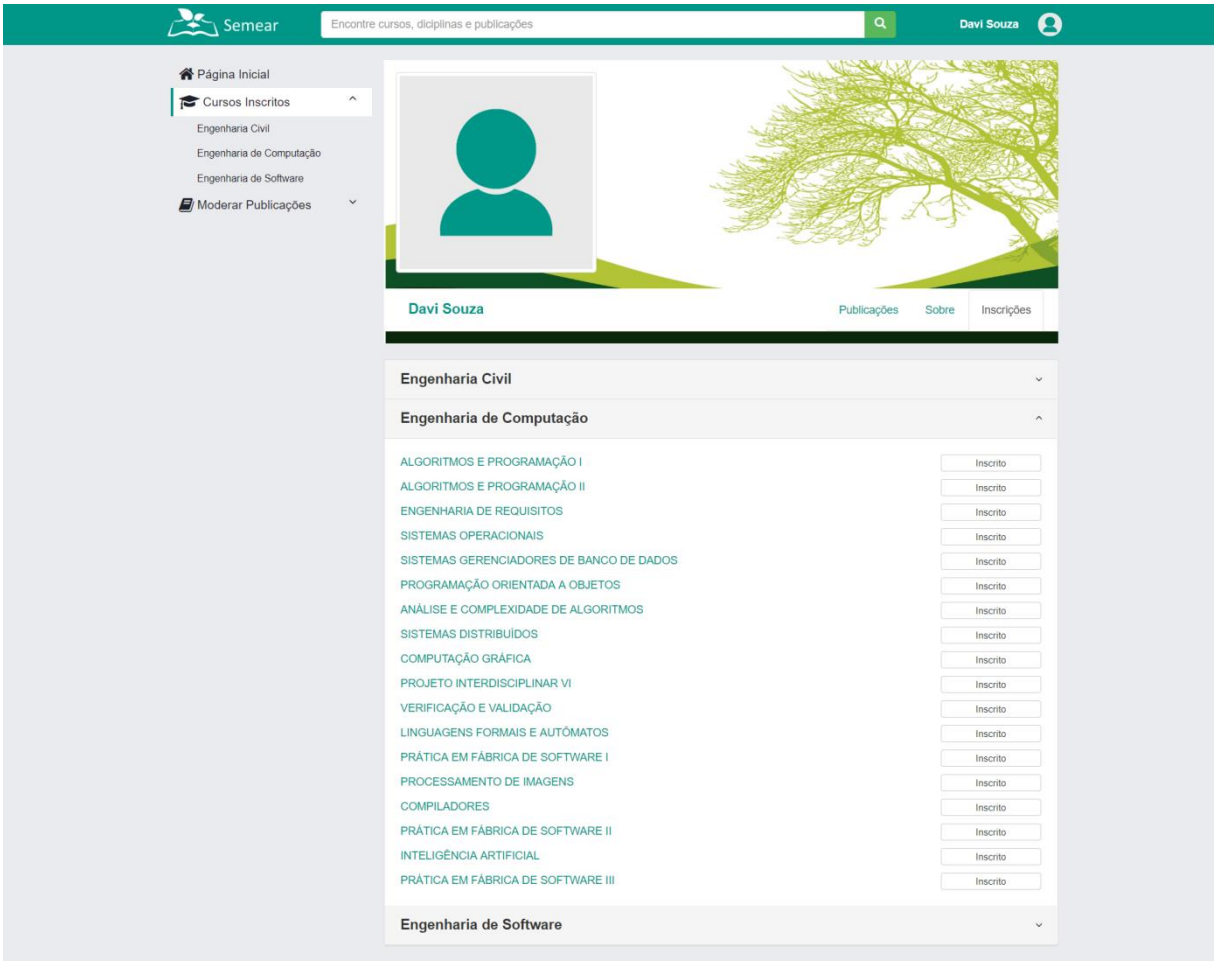
Perfil Aluno - Publicações



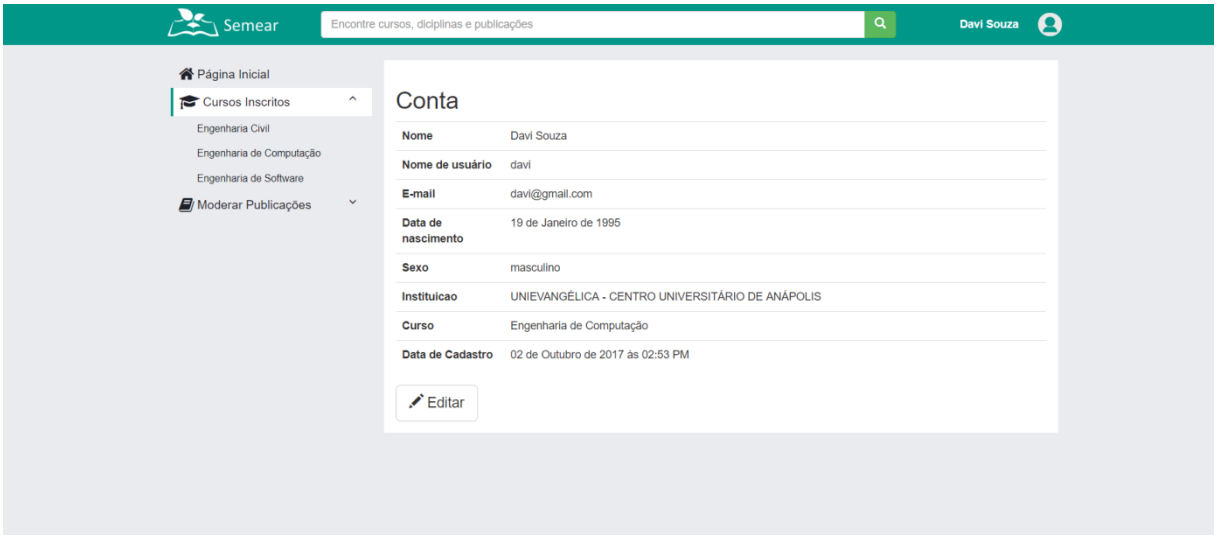
Perfil Aluno - Sobre



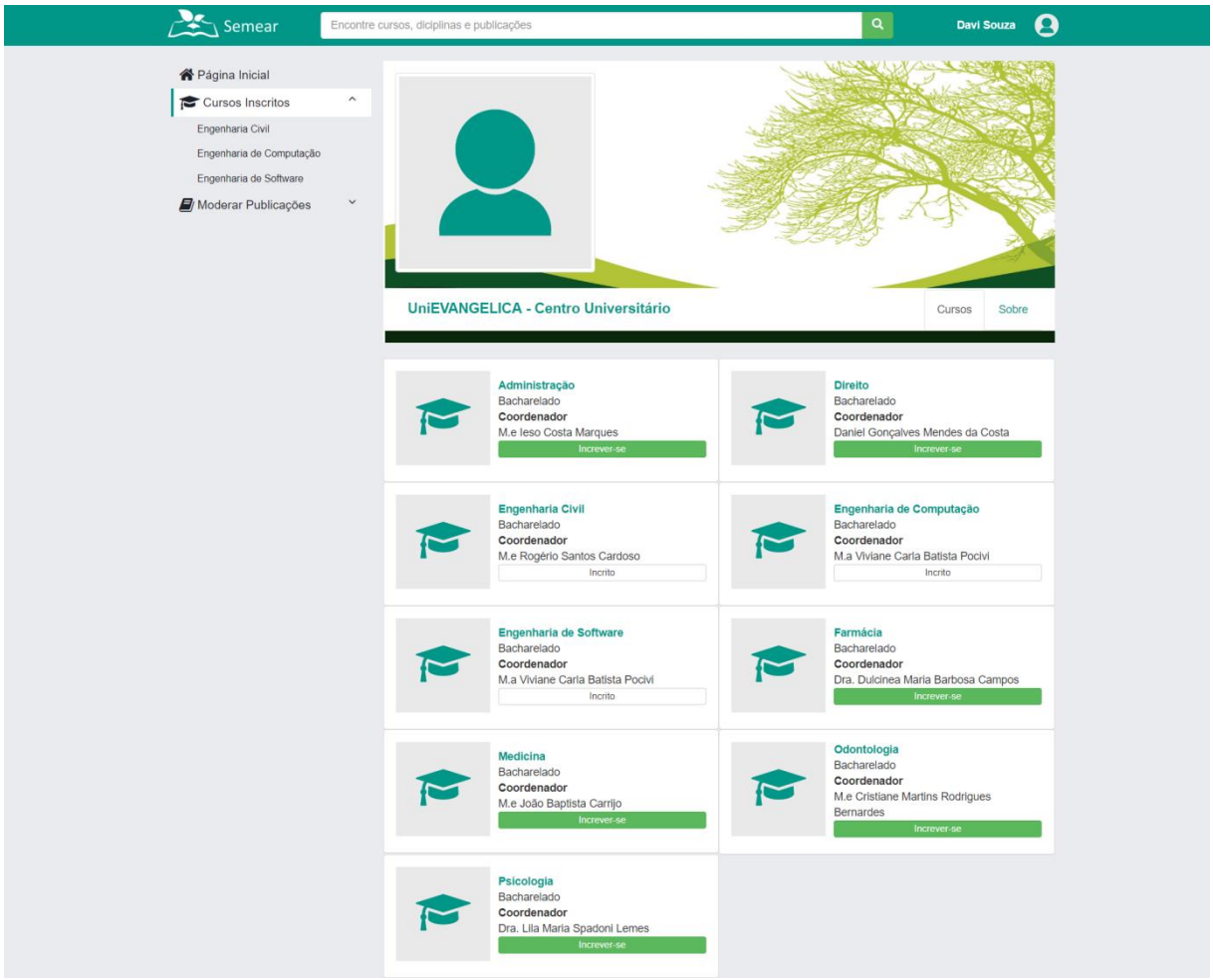
Perfil Aluno - Inscrições



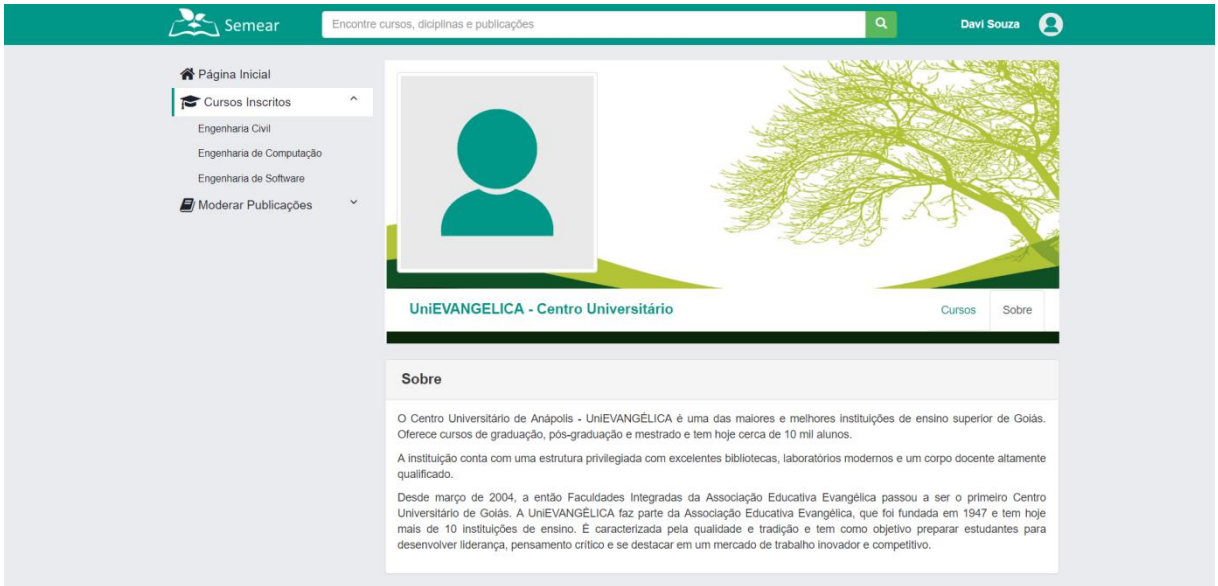
Perfil Aluno - Conta



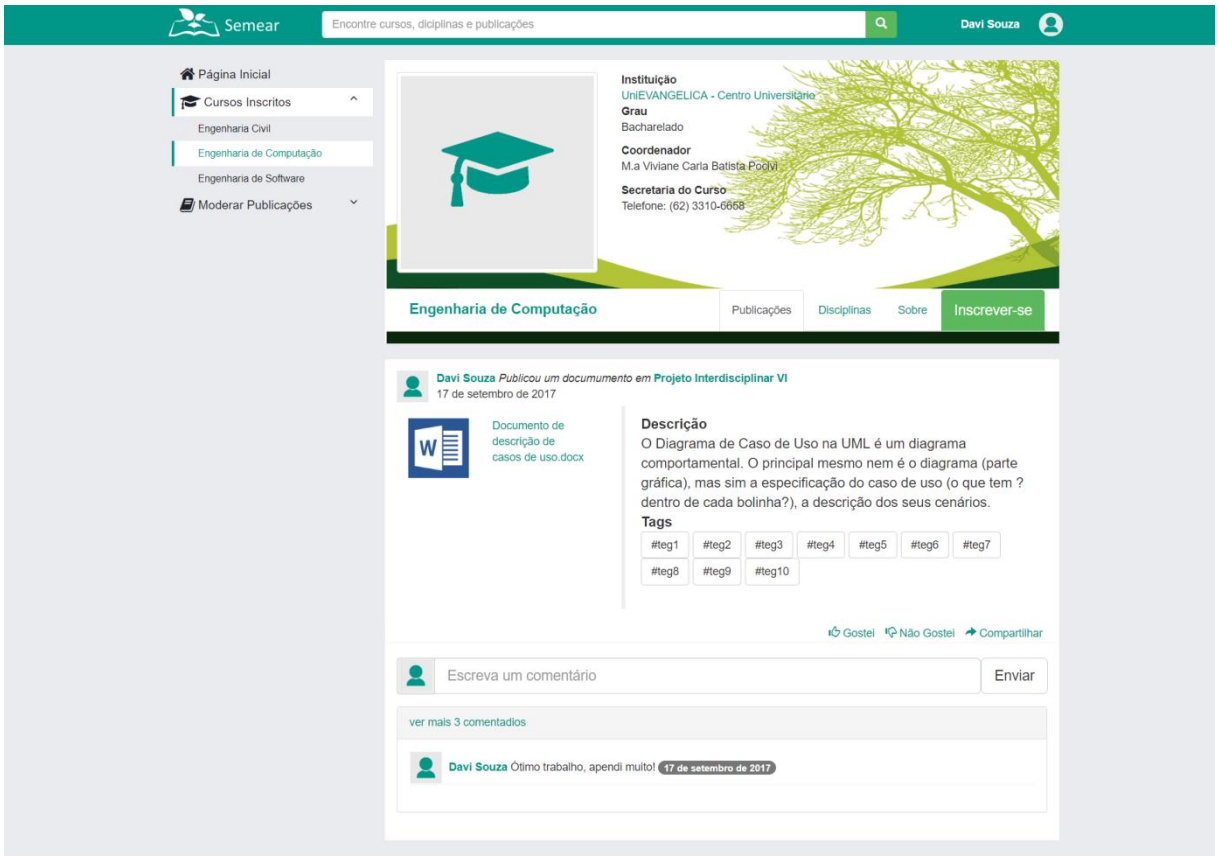
Perfil Instituição - Cursos



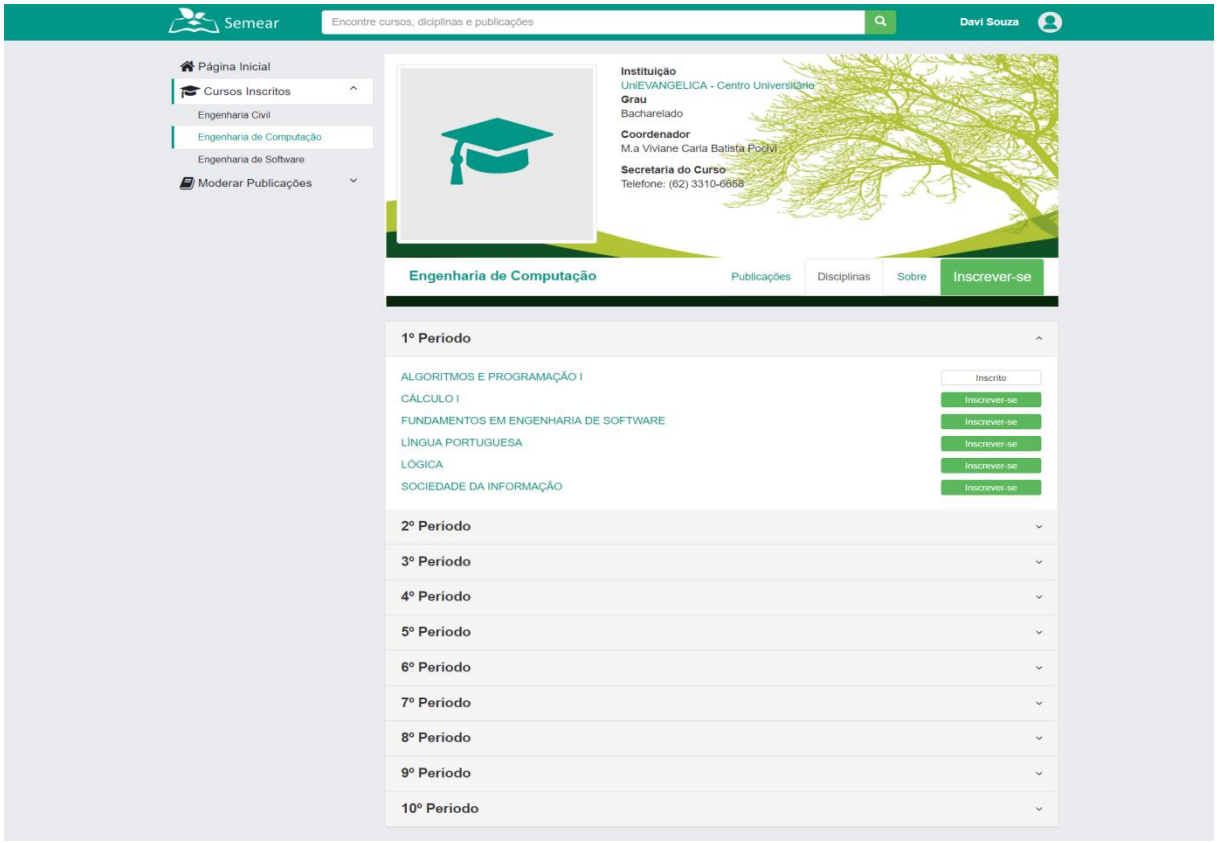
Perfil Instituição - Sobre



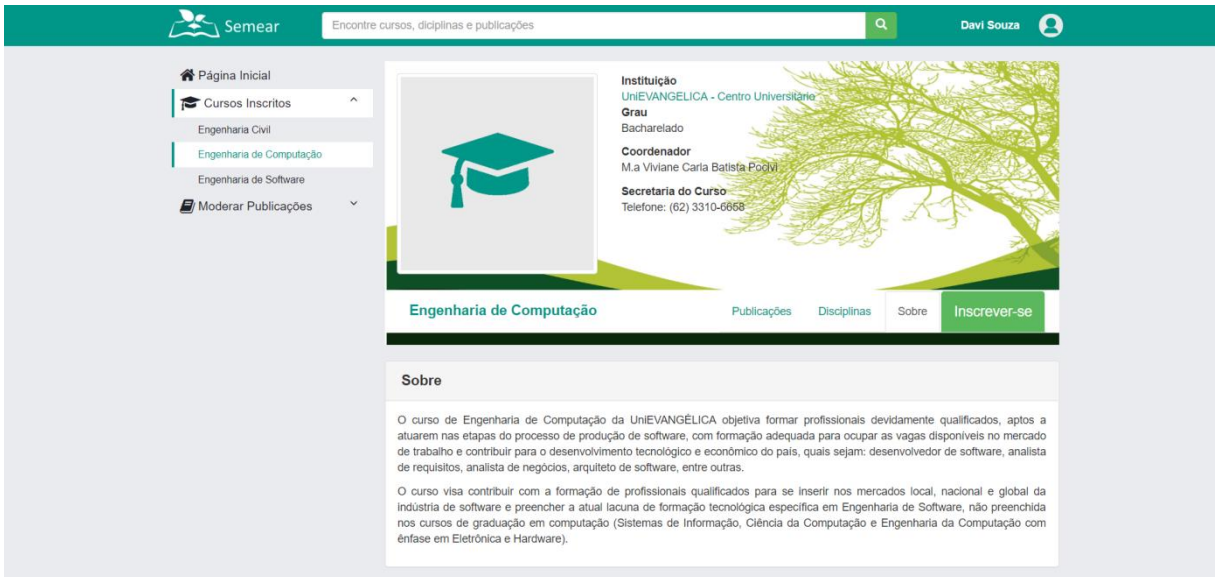
Perfil Curso - Publicações



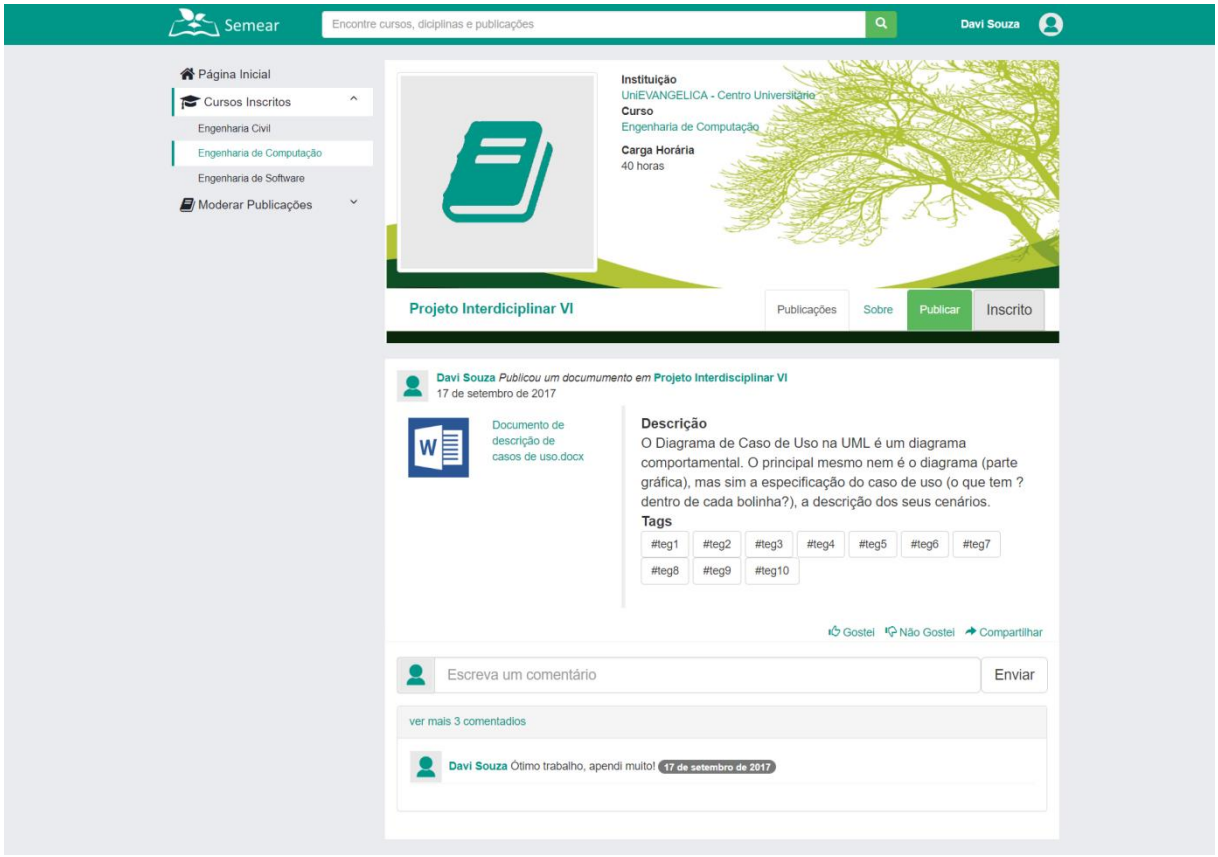
Perfil Curso - Disciplinas



Perfil Curso - Sobre

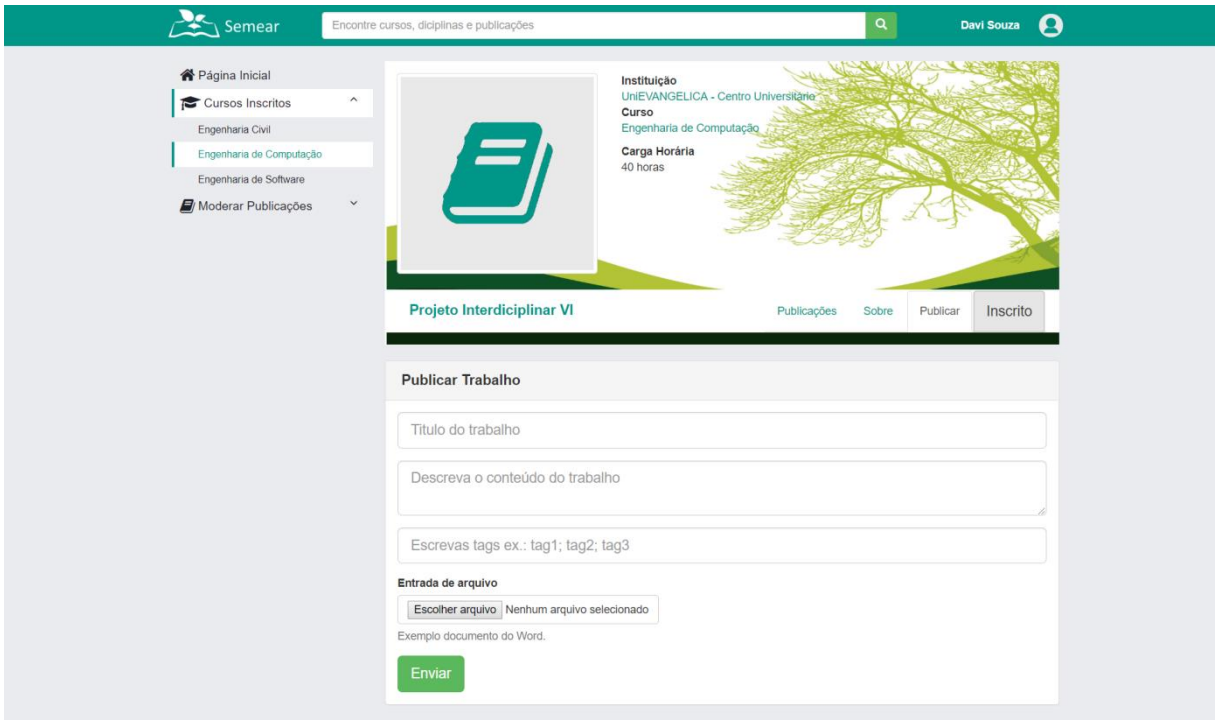


Perfil Disciplina - Publicações

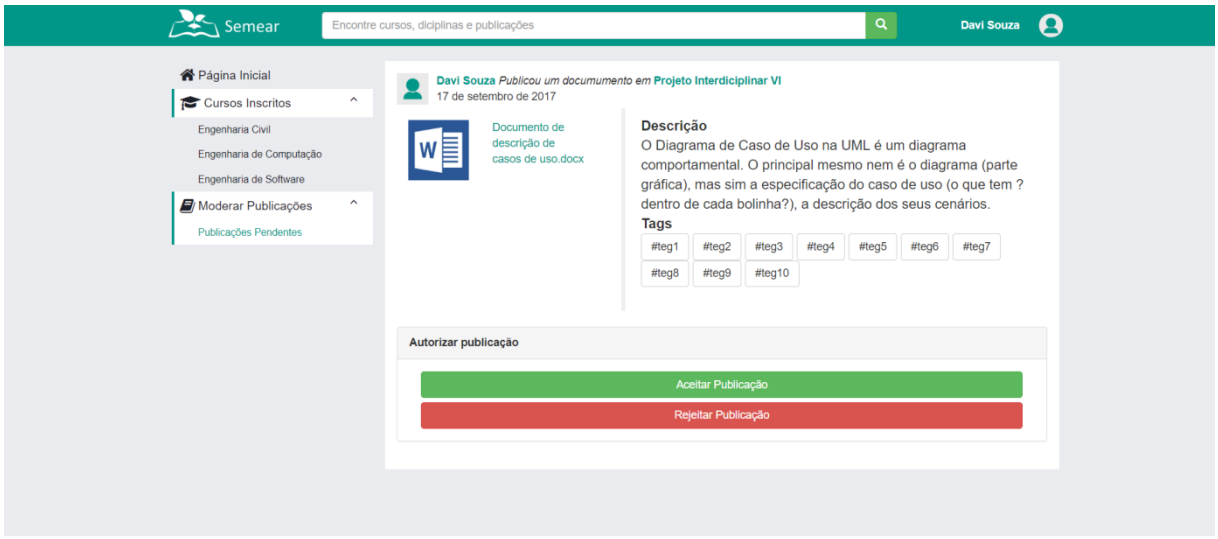


Perfil Disciplina - Sobre

Perfil Disciplina - Publicar



Moderar Publicações



APÊNDICE D – HISTÓRIAS DE USUÁRIO

História de Usuário: 002 – Manter Cadastro
Versão: 0.1
Modificação: 25/10/2019

1. VISÃO GERAL DO PROCESSO

Por meio deste requisito é possível que qualquer usuário com acesso a página possa realizar o cadastro na plataforma.

2. ATORES/PERFIS DO PROCESSO

NOME DO PERFIL	TIPO DE PERFIL	DESCRIÇÃO DO PERFIL
Administrador	☐ Interno ☒ Externo	Qualquer usuário com acesso a plataforma.

3. COMO DEMONSTRAR

Página principal -> formulário Detalhes da Conta

4. HISTÓRIA DE USUÁRIO

HISTÓRIA DE USUÁRIO	DESCRIÇÃO	STATUS	PRIORIDADE
UH. 002	Manter Cadastro	Em desenvolvimento	Essencial
Eu como: Usuário.			
Quero: Realizar o cadastro de minhas informações no sistema			
Para: Que eu possa ter acesso as informações e funcionalidades presentes no sistema.			

CRITÉRIOS DE ACEITAÇÃO	
ITEM	**DETALHAMENTO**
1	O sistema apresenta a tela “Inicial”.
2	Usuário informa corretamente todos os dados do formulário.
3	Usuário aciona o botão “Criar Conta”.
3.1	Redireciona para a tela “Lista de Cursos”.

5. FLUXO DE EXCEÇÃO

FLUXO DE EXCEÇÃO	
ITEM	**DETALHAMENTO**
1	Caso os dados informados sejam inválidos, apresentar as mensagens abaixo do próprio campo.
1.1	Caso os campos obrigatórios estejam em branco, deve-se apresentar a mensagem “O <nome do campo> é obrigatório.”.
1.2	Caso o usuário já esteja cadastrado na base de dados, apresentar a mensagem " usuário já cadastrado. ".

6. LINKS E BOTÕES

NOME	TIPO	FUNÇÃO	REGRA
Criar Conta	Botão	Verificar a informação digitada no formulário e persiste na base de dados.	Ao submeter o formulário, desabilitar o botão. Em caso de erro, habilitar o botão.

7. CAMPOS

NOME	TAMANHO	OBRIGATÓRIO	TIPO	DESCRIÇÃO
Nome	100	☒ Sim ☐ Não	Texto	Não se aplica
Sobrenome	100	☒ Sim ☐ Não	Texto	Não se aplica
Instituição	-	Sim ☒ Não	Select	Não se aplica
Curso	-	☐ Sim ☒ Não	Select	Não se aplica
email	100	☒ Sim ☐ Não	email	Não se aplica
senha	-	☒ Sim ☐ Não	Password	Não se aplica

Dia	02	☐ Sim ☐ Não	number	Não se aplica
Mes	-	☒ Sim ☐ Não	Select	Não se aplica
Ano	4	☒ Sim ☐ Não	number	Não se aplica
Masculino; Feminino; Não Declarar.	-	☒ Sim ☐ Não	radio	Não se aplica

História de Usuário: 009 – Upload Trabalhos
Versão: 0.1
Modificação: 25/10/2019

1. VISÃO GERAL DO PROCESSO

Por meio deste requisito é possível que qualquer usuário cadastrado possa realizar o upload de trabalhos na plataforma.

2. ATORES/PERFIS DO PROCESSO

NOME DO PERFIL	TIPO DE PERFIL	DESCRIÇÃO DO PERFIL
Qualquer usuário atutenticado	☒ Interno ☐ Externo	Qualquer usuário autenticado no sistema.

3. COMO DEMONSTRAR

Menu principal -> Novo Post

4. HISTÓRIA DE USUÁRIO

HISTÓRIA DE USUÁRIO	DESCRIÇÃO	STATUS	PRIORIDADE
UH. 009	Upload de Trabalhos	Em desenvolvimento	Essencial
Eu como: Usuário.			
Quero: Realizar o upload de trabalhos no sistema			
Para: Que outros usuários cadastrados possam visualizar o anexo enviado.			

CRITÉRIOS DE ACEITAÇÃO	
ITEM	DETALHAMENTO
1	O sistema apresenta a tela “Inicial”.
2	Usuário informa corretamente todos os dados do formulário.
3	Usuário aciona o botão “Escolher Arquivo”.
3.1	O sistema retorna a mensagem de sucesso.

5. FLUXO DE EXCEÇÃO

FLUXO DE EXCEÇÃO

ITEM	DETALHAMENTO
1	Caso os dados informados sejam inválidos, apresentar as mensagens abaixo do próprio campo.
1.1	Caso os campos obrigatórios estejam em branco, deve-se apresentar a mensagem “O <nome do campo> é obrigatório.”.

6. LINKS E BOTÕES

NOME	TIPO	FUNÇÃO	REGRA
Escolher Arquivo	Botão	Verificar a informação digitada no formulário e persiste na base de dados.	Ao submeter o formulário, desabilitar o botão. Em caso de erro, habilitar o botão.

História de Usuário: 010 – Dados do Artigo
Versão: 0.1
Modificação: 25/10/2019

1. VISÃO GERAL DO PROCESSO

Por meio deste requisito é possível que qualquer usuário cadastrado possa realizar o cadastro de informações de uma publicação no sistema.

2. ATORES/PERFIS DO PROCESSO

NOME DO PERFIL	TIPO DE PERFIL	DESCRIÇÃO DO PERFIL
Qualquer usuário atutenticado	☒ Interno ☐ Externo	Qualquer usuário autenticado no sistema.

3. COMO DEMONSTRAR

Menu principal -> Novo Post

4. HISTÓRIA DE USUÁRIO

HISTÓRIA DE USUÁRIO	DESCRIÇÃO	STATUS	PRIORIDADE
UH. 009	Inserir descrição do que será tratado no artigo	Em desenvolvimento	Essencial
Eu como: Usuário.			
Quero: Realizar o cadastro de informações sobre uma determinada publicação no sistema			
Para: Que outros usuários cadastrados possam visualizar os dados gravados na publicação.			

CRITÉRIOS DE ACEITAÇÃO	
ITEM	**DETALHAMENTO**
1	O sistema apresenta a tela “Inicial”.
2	Usuário informa corretamente todos os dados do formulário.
3	Usuário aciona o botão “Salvar”.
3.1	O sistema retorna a mensagem de sucesso.

5. FLUXO DE EXCEÇÃO

FLUXO DE EXCEÇÃO	
ITEM	**DETALHAMENTO**
1	Caso os dados informados sejam inválidos, apresentar as mensagens abaixo do próprio campo.
1.1	Caso os campos obrigatórios estejam em branco, deve-se apresentar a mensagem “O <nome do campo> é obrigatório.”.

6. LINKS E BOTÕES

NOME	TIPO	FUNÇÃO	REGRA
Escolher Arquivo	Botão	Verificar a informação digitada no formulário e persiste na base de dados.	Ao submeter o formulário, desabilitar o botão. Em caso de erro, habilitar o botão.

7. CAMPOS

NOME	TAMANHO	OBRIGATÓRIO	TIPO	DESCRIÇÃO
Título do Trabalho	100	☒ Sim ☐ Não	Texto	Não se aplica
Curso	-	☒ Sim ☐ Não	Select	Não se aplica
Resumo do Artigo	500	☒ Sim ☐ Não	Text	Não se aplica

História de Usuário: 015 – Cadastrar Curso
Versão: 0.1
Modificação: 21/08/2019

1. VISÃO GERAL DO PROCESSO

Por meio deste requisito é possível que o administrador cadastre um novo curso no SEMEAR.

2. ATORES/PERFIS DO PROCESSO

NOME DO PERFIL	TIPO DE PERFIL	DESCRIÇÃO DO PERFIL
Administrador	☒ Interno ☐ Externo	Usuário autenticado com perfil de administrador.

3. COMO DEMONSTRAR

Acessar o menu principal -> Cadastrar Curso

4. HISTÓRIA DE USUÁRIO

HISTÓRIA DE USUÁRIO	DESCRIÇÃO	STATUS	PRIORIDADE
UH. 010	Cadastrar Curso	A Desenvolver	Essencial
Eu como: Administrador.			
Quero: Realizar o cadastro de um curso no sistema.			
Para: Que todos os usuários autenticados possam visualizar informações a respeito do curso cadastrado.			

CRITÉRIOS DE ACEITAÇÃO	
ITEM	**DETALHAMENTO**
1	O sistema apresenta a tela “Cadastrar Curso”.
2	Usuário informa corretamente todos os dados do formulário.
3	Usuário aciona o botão “Salvar”.
3.1	Redireciona para a tela “Instituição”.

5. FLUXO DE EXCEÇÃO

FLUXO DE EXCEÇÃO	
ITEM	**DETALHAMENTO**
1	Caso os dados informados sejam inválidos, apresentar as mensagens abaixo do próprio campo.
1.1	Caso os campos obrigatórios estejam em branco, deve-se apresentar a mensagem “O <nome do campo> é obrigatório.”.
1.2	Caso o curso já esteja cadastrado na base de dados, apresentar a mensagem " curso já cadastrado. ".

6. LINKS E BOTÕES

NOME	TIPO	FUNÇÃO	REGRA
Cadastrar	Botão	Verificar a informação digitada no formulário e persiste na base de dados.	Ao submeter o formulário, desabilitar o botão. Em caso de erro, habilitar o botão.

7. CAMPOS

NOME	TAMANHO	OBRIGATÓRIO	TIPO	DESCRIÇÃO
Curso	50	☒ Sim ☐ Não	Texto	Não se aplica
Instituição	-	☒ Sim ☐ Não	Select	Não se aplica
Grau	-	☒ Sim Não	Texto	Não se aplica
Modalidade	50	☒ Sim ☐ Não	Texto	Não se aplica
Campus	100	☒ Sim ☐ Não	Texto	Não se aplica

Turno	-	☒ Sim ☐ Não	Select	Não se aplica
Sobre o Curso	500	☐ Sim ☒ Não	Texto	Não se aplica

História de Usuário: 016 – Realizar Login
Versão: 0.1
Modificação: 25/10/2019

1. VISÃO GERAL DO PROCESSO

Por meio deste requisito é possível que qualquer usuário cadastrado possa realizar o cadastro de informações de uma publicação no sistema.

2. ATORES/PERFIS DO PROCESSO

NOME DO PERFIL	TIPO DE PERFIL	DESCRIÇÃO DO PERFIL
Qualquer usuário com conta na plataforma	☐ Interno ☒ Externo	Qualquer usuário autenticado no sistema.

3. COMO DEMONSTRAR

Menu principal -> Login

4. HISTÓRIA DE USUÁRIO

HISTÓRIA DE USUÁRIO	DESCRIÇÃO	STATUS	PRIORIDADE
UH. 016	Realizar login no sistema	Concluído	Essencial
Eu como: Usuário.			
Quero: Realizar login no sistema.			
Para: Que eu possa ter acesso as funcionalidades e informações presentes no sistema.			

CRITÉRIOS DE ACEITAÇÃO	
ITEM	DETALHAMENTO
1	O sistema apresenta a tela “Inicial”.
2	Usuário informa corretamente todos os dados do formulário.
3	Usuário aciona o botão “Entrar”.
3.1	O sistema redireciona o usuário para a tela da Instituição

5. FLUXO DE EXCEÇÃO

FLUXO DE EXCEÇÃO	
ITEM	**DETALHAMENTO**
1	Caso os dados informados sejam inválidos, apresentar as mensagens abaixo do próprio campo.
1.1	Caso os campos obrigatórios estejam em branco, deve-se apresentar a mensagem “O <nome do campo> é obrigatório.”.

6. LINKS E BOTÕES

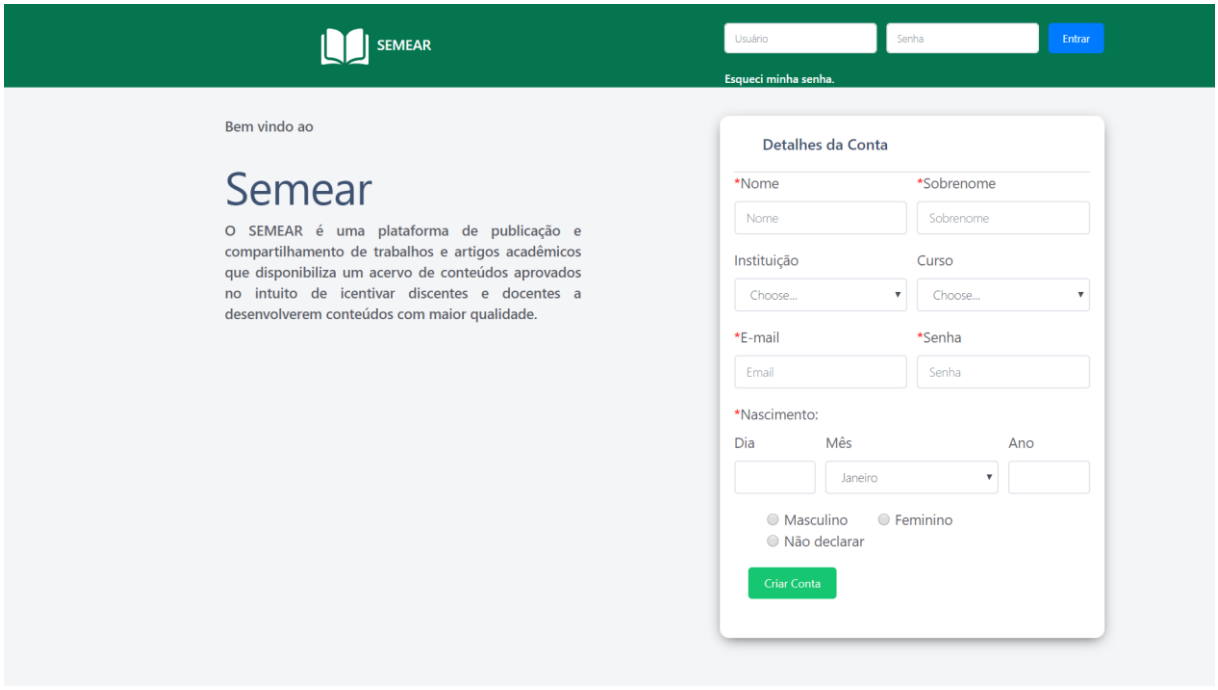
NOME	TIPO	FUNÇÃO	REGRA
Entrar	Botão	Verificar a informação digitada é válida.	Ao submeter o formulário, desabilitar o botão. Em caso de erro, habilitar o botão.

7. CAMPOS

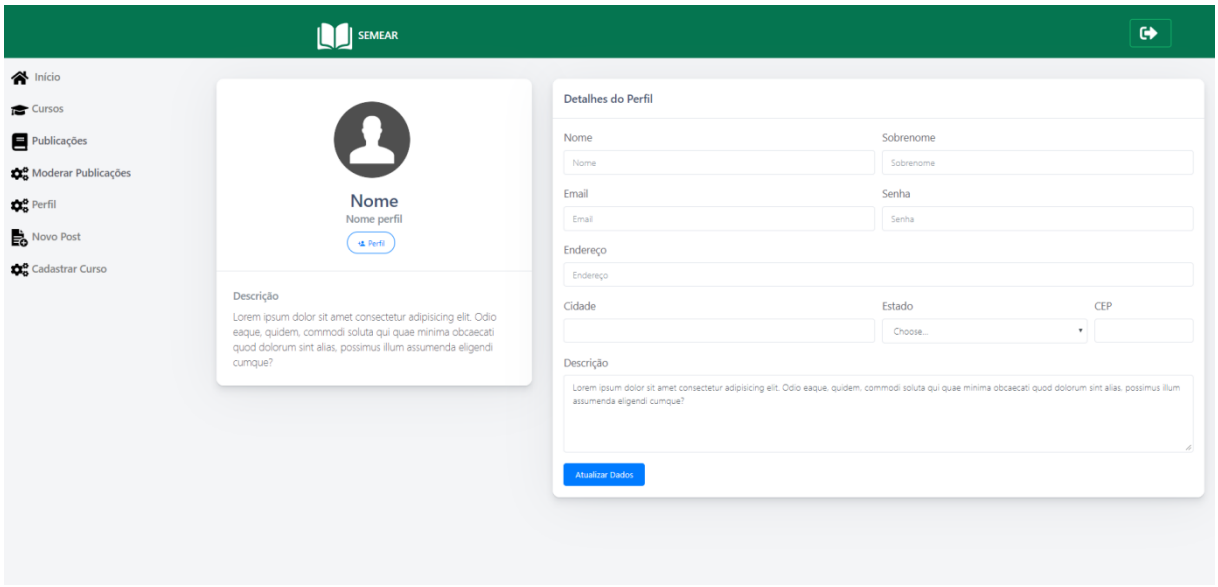
NOME	TAMANHO	OBRIGATÓRIO	TIPO	DESCRIÇÃO
Usuário	100	☒ Sim ☐ Não	Texto	Não se aplica
senha	-	☒ Sim ☐ Não	password	Não se aplica

APÊNDICE E – TELAS DO SISTEMA

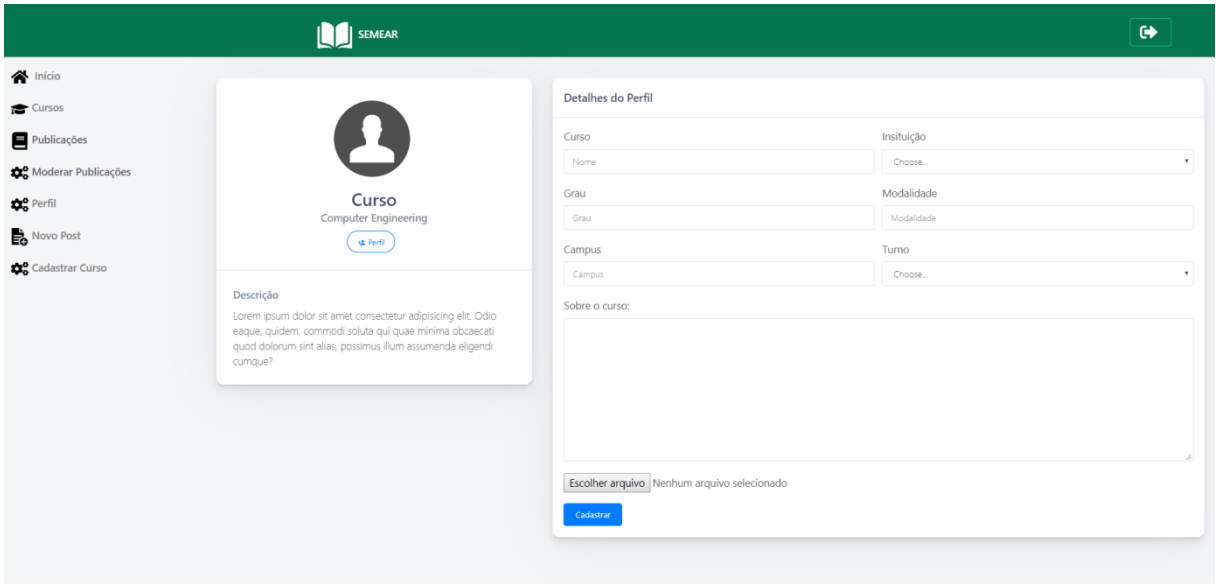
Login e Cadastro Sistema



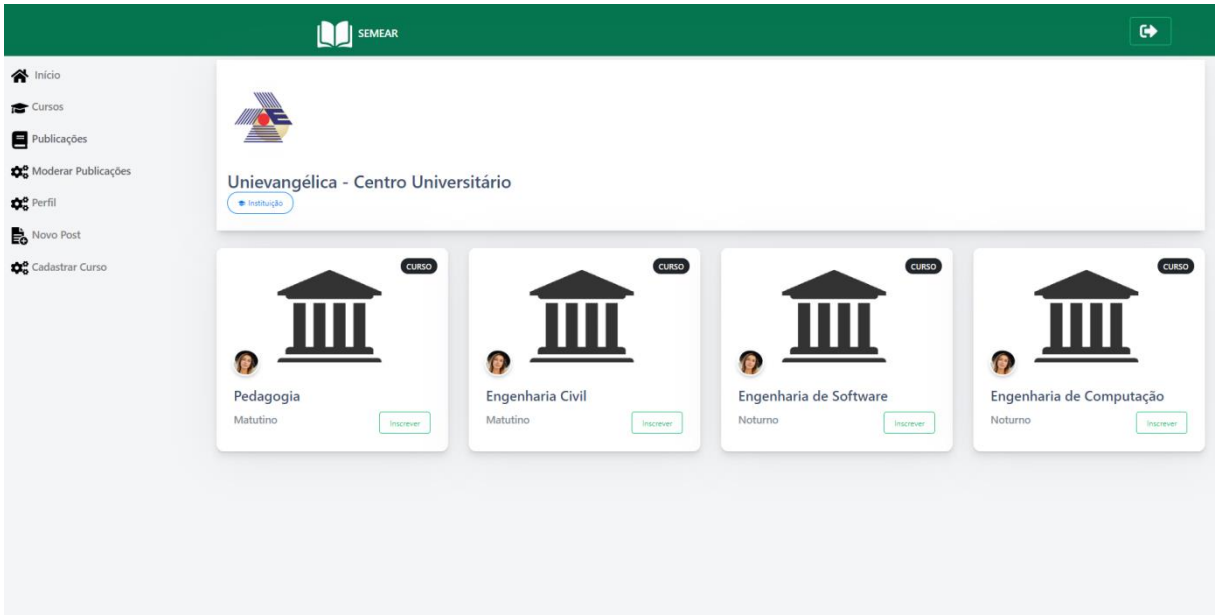
Alterar Cadastro Usuário



Cadastro Curso



Lista de cursos cadastrados



Upload de trabalhos e Dados Artigo