SGMS - SISTEMA GERENCIADOR DE MUDANÇAS DE SOFTWARE

BATALHA, Thiago M.[1]; CAMPOS, Saulo C.[2];
BAIA, Joás Weslei[2]; TREVIZANO, Waldir Andrade[2]

[1] Graduando do curso de Ciência da Computação – UNIFAGOC
[2] Docente do curso de Ciência da Computação UNIFAGOC

thiagomoreira60@gmail.com
saulo@unifagoc.edu.br
joas.baia@unifagoc.edu.br
waldir@unifagoc.edu.br

RESUMO
Este trabalho visa propor a adoção de um modelo de gestão de mudanças para a empresa Cucabrasil por meio de uma aplicação web criada ao longo do projeto. Fundamentada em boas práticas da engenharia de software e amparada pelo modelo de gestão de mudanças proposto pela ITIL, a ferramenta tem por objetivo auxiliar a empresa no processo de gestão de mudanças de seus softwares, solicitadas pelos seus clientes. Para o desenvolvimento do projeto foram utilizadas tecnologias atuais que estão presentes no mercado de desenvolvimento de softwares.

Palavras-chave: Gerência de mudanças. Software. Serviços de TI. ITIL. Engenharia de Software.

INTRODUÇÃO

Um software é um conjunto de instruções escritas por quem o desenvolveu que serão executadas em dispositivos eletrônicos capazes de entendê-las e tentar realizar as ações descritas nelas (Gogoni, 2019). Os softwares são utilizados em computadores, aparelhos celulares, televisões, rádios e em outros tipos de dispositivos eletrônicos, com a finalidade de realizar diversas operações, como processos de automação, realização de operações matemáticas, edição de imagens e muitos outros, tendo como objetivo o auxílio ou até mesmo a substituição do trabalho humano na realização de tarefas, em resumo, a resolução de problemas.

Para o desenvolvimento de um software, é necessária a passagem por algumas etapas, como o conhecimento do problema a se resolver, o levantamento de requisitos, a modelagem, a codificação, testes e a sua manutenção até o final do seu ciclo de vida. Esses processos precisam ser realizados de forma segura e objetiva, visando à entrega de um software funcional (Sommerville, 2019).

Sommerville (2019) explica sobre a dinâmica da evolução dos sistemas de software através das leis de Lehman, pioneiro sobre o assunto nos anos 70, que, segundo ele, são aplicadas em todos os sistemas de software de larga escala:

1. Um programa deve mudar continuamente para continuar sendo útil;
2. À medida que um programa em desenvolvimento muda, a sua estrutura se degrada;
3. Ao longo da vida útil de um programa, a taxa de mudança é aproximadamente constante e independente dos recursos disponíveis;
4. A mudança incremental em cada lançamento de um sistema é aproximadamente constante;

5. Novas funcionalidades devem ser acrescentadas ao sistema para aumentar a satisfação do usuário.

Na engenharia de software, há uma subárea específica para tratar da gestão de mudanças. Conforme Sommerville (2019), o processo de gestão de mudanças é importante, pois, a partir dele, é possível avaliar o impacto, identificar os componentes afetados e definir o custo da mudança. Ainda sobre mudanças de software, Sommerville (2019) menciona a existência de três tipos: "corretiva", para realizar reparos de funcionalidades defeituosas; "adaptativa", para adaptar o software a novas plataformas e ambientes; e, por fim, "perfectiva", ou evolutiva, que trata de alterações para aperfeiçoar o software, implementando novas funcionalidades.

A Biblioteca ITIL (*Information Technology Infrastructure Library*), definida como uma metodologia que sugere uma série de boas práticas sobre a gestão de serviços de TI, também aborda a gestão de mudanças (Freitas, 2011). Assim, para agregar ainda mais valor ao processo, pode-se introduzir os conceitos da ITIL com o objetivo de assegurar a eficácia na gestão de seus processos e uma boa experiência para os seus usuários (Silva, 2021).

A empresa Cucabrasil, situada na cidade de Ubá, Minas Gerais, desde o ano de 2008 trabalha com o desenvolvimento de softwares para atender empresas de pequeno e médio porte dos segmentos industrial e comercial de sua cidade e região. Colaborando com seus clientes, os sistemas da Cucabrasil auxiliam no gerenciamento das empresas em diversos setores, integrando a gestão dessas áreas em um só lugar, automatizando processos, disponibilizando e provendo informações detalhadas e importantes para que os usuários dos sistemas tenham um melhor conhecimento do estado de saúde de sua empresa, dessa forma, facilitando o processo de tomada de decisões.

Os softwares produzidos pela Cucabrasil estão em constante evolução e passam por mudanças frequentemente, seja por demandas de seus clientes ou demandas externas, como regulamentações governamentais, exigências do meio de negócio, evolução tecnológica, entre outras. Tais mudanças, quando identificadas, precisam ser analisadas e ordenadas segundo os critérios de avaliação de prioridade da empresa para serem desenvolvidas e entregues aos clientes, gerando o devido valor.

Até a conclusão deste trabalho, a empresa não conta com uma ferramenta que auxilie no gerenciamento das mudanças de forma apropriada, segundo os conceitos da engenharia de software. Todas as solicitações são armazenadas em um "bloco de notas" contendo apenas o solicitante e uma breve descrição da mudança. Essa prática ocasiona diversos problemas, como: impossibilidade de uma definição clara de prioridades, dificuldade para selecionar as mudanças a serem trabalhadas, leitura extensa de todo arquivo para encontrar detalhes simples da mudança. Ademais, sempre que alguém está acessando o arquivo, deve-se garantir que outra pessoa não o acesse para prevenir a sobrescrita de alterações realizadas durante o processo.

O problema abordado neste trabalho é: como melhorar a qualidade do processo de mudanças de software na Cuca Brasil? Como objetivo geral, este trabalho propõe a elaboração de um processo de gerência de mudanças, inspirado no processo proposto pela ITIL e apoiado por uma ferramenta de software, desenvolvida e construída durante o processo.

Com a implementação do processo e a utilização do software, espera-se que a empresa passe a ter a documentação das solicitações de mudança, com um maior nível de detalhamento, uma melhor capacidade de análise das mudanças e a redução do índice de falhas e retrabalhos.

REFERENCIAL TEÓRICO

Biblioteca ITIL

A Biblioteca ITIL (*Information Technology Infrastructure Library*), em sua tradução literal, significa Biblioteca de Infraestrutura e Tecnologia da Informação. Freitas (2011) diz em seu livro que, apesar de sua tradução apresentar a nomenclatura de biblioteca, esse termo é mais utilizado quando se deseja referir ao conjunto de livros que abriga as recomendações e práticas sobre o tema; contudo, popularmente, a ITIL é conhecida como um conjunto de recomendações baseadas em boas práticas de Gerenciamento de Serviços de TI, referido na maioria das vezes como "O ITIL".

O ITIL atualmente está em sua 4ª versão, lançada no ano de 2019; contudo, segundo Freitas (2011), tudo começou na década de 80, no Reino Unido, quando a Agência Central de Computadores e Telecomunicações desenvolveu o Método de Governo de Infraestrutura de Tecnologia da Informação (GITIM), que, posteriormente, mais precisamente na década de 89, foi renomeado para ITIL, surgindo assim sua primeira versão, distribuída em 31 livros. Com o decorrer dos anos e o surgimento das novas versões do ITIL, alguns conhecimentos foram organizados e simplificados, fazendo com que o conteúdo dos originalmente 31 livros, na versão 1, fossem reduzidos a7 livros na versão 2, lançada por volta do ano 2000, e depois para 5 livros na versão 3, lançada entre 2007 e 2008 (Freitas, 2011). A lista a seguir demonstra como ficaram divididos os temas na biblioteca do ITIL na versão 3, que representa o ciclo de vida de um serviço de TI:

- *Service Strategy* (Estratégia de Serviço);
- *Service Design* (Desenho de Serviço);
- *Service Transition* (Transição de Serviço);
- *Service Operations* (Operação de Serviço);
- *Continual Service Improvement* (Melhoria Continuada de Serviço).

Service Transition (Transição de Serviço)

Freitas (2011, p. 249), em seu livro Fundamentos do Gerenciamento de Serviços de TI, define que a responsabilidade da etapa de Transição de Serviço é "... garantir que serviços novos, modificados ou removidos atendam às necessidades do negócio de acordo com os ciclos de Estratégia e Desenho de Serviço". Entre diversos processos abordados na Transição de Serviços, encontra-se o processo de gerência de mudança, o foco deste trabalho.

Segundo Freitas (2011), o ITIL propõe que toda mudança deve ser passada por uma análise de impacto precisa, na qual devem ser observados alguns detalhes sobre o processo de mudanças. Esses detalhes estão contidos em uma política na qual deve-se responder a 7 questões, conhecidas como "7Rs":

- *RAISED* (levantou, solicitou): quem REQUISITOU a mudança?
- *REASON* (razão): qual é a RAZÃO para a mudança?
- RETURN (retorno): qual é o RETORNO requerido para a mudança?

- *RISKS* (riscos): quais são os RISCOS envolvidos na mudança?
- *RESOURCES* (recursos): que RECURSOS são requeridos para entregar a mudança?
- *RESPONSIBLE* (responsável): quem é RESPONSÁVEL por construir, testar e implementar a mudança?
- *RELATIONSHIP* (relacionamento): qual é o RELACIONAMENTO entre essa mudança e outras mudanças?

METODOLOGIA

Para que este projeto pudesse ser desenvolvido, foram inicialmente realizadas pesquisas buscando conhecimento sobre boas práticas para aplicar sob a gerência das mudanças solicitadas pelos clientes da Cucabrasil, podendo assim convertê-lo em uma solução tecnológica para atender à necessidade da empresa. Através dessa pesquisa, encontrou-se a biblioteca ITIL, capaz de contribuir com suas boas práticas de serviços de TI, mais especificamente sob a etapa de transição de serviço. Livros, sites, vídeos e diversos artigos foram consultados com objetivo de se obter o conhecimento sobre o *framework* para ajudar no desenvolvimento do protótipo do software e auxiliar na gerência das mudanças, desde a sua concepção até a sua entrega.

Com o conceito do ITIL fixado, foram realizadas algumas reuniões com os analistas de sistemas da empresa Cucabrasil para entender as necessidades da empresa e definir como seria realizado o processo de gerenciamento da mudança. Após o desenho do processo, iniciou-se o desenvolvimento do software para apoio e gerência.

Para o desenvolvimento do software, optou-se pela plataforma de desenvolvimento WEB pela razão de ser suportada nos principais sistemas operacionais como Windows, Linux, IOS, Android, entre outros, a partir do mesmo código-fonte.

Para a construção do front-end, além das tecnologias básicas para desenvolvimento de páginas web (HTML, CSS e Javascript), através dos frameworks *Bootstrap* (versão 5.2), para estilização, e o EJS *(Embedded JavaScript templating)*, para construção dos elementos. A própria mantenedora do *Bootstrap* (2002) o define como "*kit de ferramentas de front-end poderoso, extensível e repleto de recursos*", capaz prover um várias estilizações de cores, fontes, posicionamento, tamanhos e diversos componentes já prontos para a utilização. Já o
EJS é um *view-engine*, utilizado para gerar marcações HTML a partir de Js simples (EJS, 2022). Ele é apelidado por seus usuários como "*um HTML com superpoderes*", pois essa ferramenta permite a utilização de objetos, variáveis, condicionais, laços e outros artifícios na construção de HTML com apenas algumas marcações específicas da sua sintaxe.

Para adicionar mais riqueza de recursos às páginas web da aplicação, principalmente em relação à visualização de dados em tabelas, utilizou-se o *framework* JS Datatables, que em seu site (Datatables, 2022) é definido como "*plug-in para a biblioteca jQuery Javascript, uma ferramenta altamente flexível, que adiciona recursos avançados a qualquer tabela HTML*". Essa ferramenta é responsável por aplicar diversas funcionalidades em tabelas HTML, como filtros, ordenação de registros por coluna, paginação, estilização, entre outras, facilitando o desenvolvimento ao trazer essas funcionalidades previamente implementadas.

A linguagem de programação escolhida para o *back-end* é o Javascript, utilizando o Node.js. A escolha do ambiente Node.js foi para simplificar o desenvolvimento do projeto utilizando a mesma linguagem de programação no *back-end* e *front-end*.

Para controlar as requisições HTTP, utilizou-se o *framework Express, um dos frameworks* mais utilizados no Node.js. Com ele é possível realizar o gerenciamento de requisições HTTP, integrar *view-engines* para inserir dados nos *templates*, definir algumas configurações de aplicações web, entre outros (Mozilla, 2022).

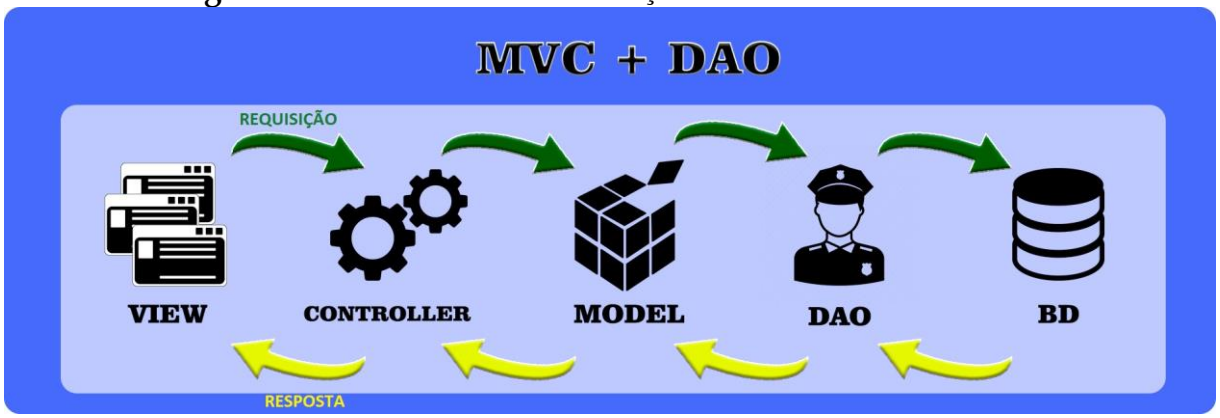
O software foi construído em uma arquitetura MVC (*Model-View-Controller*), definida por (HIGOR, 2013) como "um sistema computacional como a suas estruturas, que são compostas de elementos de software, de propriedades externamente visíveis de seus componentes e do relacionamento entre eles. Ou seja, a arquitetura define os elementos de software e como eles interagem entre si". A arquitetura foi escolhida com objetivo de diminuir o acoplamento, através da separação das responsabilidades.

Uma camada de persistência também foi adicionada ao projeto, através do padrão DAO (*Data Access Objec*t). Também visando a diminuição de acoplamento dentro do software, o objetivo desta camada é receber as requisições da camada de modelo, acessar ao banco de dados da aplicação para realizar as operações e entregar uma resposta à camada que o solicitou. Ronaldo (2015) explica que a finalidade dessa camada adicional é "separar as regras de negócios das operações referentes ao banco de dados, além de evitar que outras partes do projeto precisem preocupar-se com persistência de dados quando este não é seu principal objetivo".

Como SGBD da aplicação, foi escolhido o MySQL, um SGBD de modelo relacional mais utilizado no mundo (MYSQL, 2022). Por ser um SGBD confiável, conter recursos valiosos e conseguir processar alto volume de requisições, o MySQL compõe o pacote de ferramentas de grandes empresas como Facebook, Twitter, Booking.com, Verizon, Netflix e outros (MYSQL, 2022).

Com a utilização dessas quatro camadas, temos a estrutura de comunicação entre as camadas do software, como mostra a Figura 1.

Figura 1 - Estrutura de comunicação entre camadas do SGMS



Fonte: elaborada pelos autores.

Para auxiliar na comunicação entre a aplicação e o banco de dados, optou-se por utilizar o Knex, um *framework query-builder* para o Node.js capaz de realizar a comunicação entre uma aplicação node e alguns dos bancos de dados relacionais mais

utilizados. Seu objetivo é abstrair a utilização da linguagem SQL, linguagem de instruções para SGBDs, do desenvolvedor (KNEXJS, 2022), tornando mais simples a implementação, podendo realizar qualquer instrução no banco de dados sem que fosse digitado uma linha de SQL, apenas Js. É importante destacar que o Knex recebe instruções em Js, mas a ferramenta irá convertê-las em SQL para acessar o banco de dados. Outro benefício dessa ferramenta é a migração de SGBDs, quando necessária, por suportar os principais bancos como MySQL, PostgreSQL, MariaDB, Oracle, Firebird, entre outros, é possível a migração entre estes apenas alterando as configurações de acesso ao SGBD.

Após a definição das ferramentas e tecnologias, o processo de desenvolvimento do projeto foi realizado como apresentado na próxima sessão.

DESENVOLVIMENTO

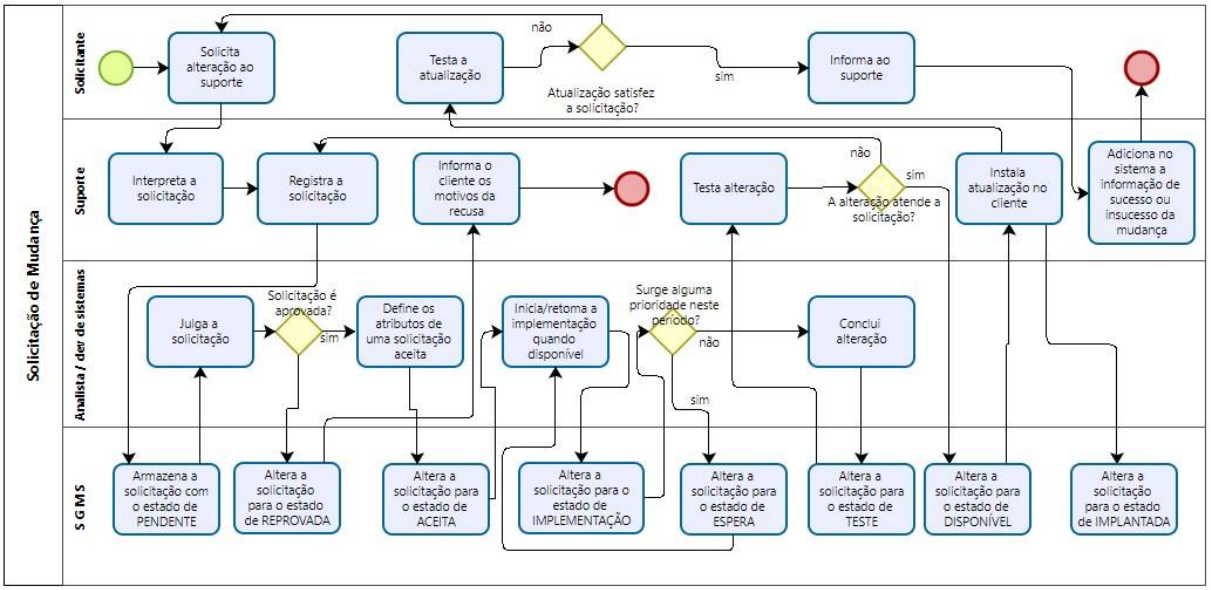
O processo de Gestão de Mudança da Cucabrasil

Como já informado, o processo de gestão de mudança foi definido a partir de reuniões realizadas com os analistas de sistemas da empresa Cucabrasil, em que foram discutidas quais seriam as melhores estratégias para o seu funcionamento, desde a solicitação efetuada pelo cliente até o *feedback* dado por ele.

Foi definido que haveria 4 atores – o solicitante, o suporte técnico, o analista/desenvolvedor e a mudança – para o processo, cuja definição ficou da seguinte forma: o solicitante reporta a mudança desejada ao suporte técnico da Cucabrasil, que registra sua solicitação. Ao ser registrada, a solicitação ficará no *status* de PENDENTE até ser julgada por um analista da empresa. Caso o analista recuse a solicitação, o suporte deverá informar ao cliente o motivo da sua rejeição; porém, se a mudança for aprovada, o analista deverá classificar o nível de prioridade e completar o formulário com informações adicionais necessárias para que uma mudança fique como ACEITA. Uma vez aceita, a mudança entrará em uma fila de acordo com sua prioridade até que seja desenvolvida. Quando o analista inicia a implementação de uma mudança, esta passa para o *status* de IMPLEMENTAÇÃO. Se, porventura, ocorrer algum imprevisto de força maior, a mudança em implementação deverá ser pausada e mudar seu *status* para EM ESPERA até que um analista possa retomá-la. Quando a implementação da mudança é concluída pelo analista, ela passa para o *status* EM TESTES para que o suporte possa testar a alteração e avaliar se está em conformidade com a solicitação do cliente. Não atendendo a solicitação, o suporte realizará um novo registro para que a mudança seja retrabalhada, do contrário, deverá instalar uma nova versão do sistema para o cliente com a mudança pronta e a mudança passará para o status IMPLANTADA. O cliente deve reportar ao suporte se a solicitação de fato atendeu às suas expectativas. Se o feedback do cliente for positivo o atendente deve finalizar o processo, caso contrário, abrir uma nova mudança para ajustes.

O processo descrito até aqui pode ser visto na Figura 2 de forma mais detalhada, através de um diagrama BPMN.

Figura 2 - Diagrama do processo de gestão de mudanças da Cucabrasil



Fonte: elaborada pelos autores

Faz parte do processo a análise e o levantamento de informações importantes para o controle e desenvolvimento das mudanças. Essas informações devem ser preenchidas pelos atores em atributos específicos durante as etapas de abertura e análise das solicitações.

Para a abertura de uma solicitação, são necessários os seguintes atributos:

- Solicitante: Responsável pela solicitação da mudança;
- Título: Descrição em poucas palavras da mudança;
- Descrição: Descrição detalhada da mudança;
- Benefícios: Apresentação prévia do impacto benéfico da mudança ao solicitante;
- Efeitos da não implementação: Possíveis impactos caso a mudança não seja implementada;
- Tipos de mudança: Correção, criação, evolução;
- Categorias: Aqui foi definida a utilização dos nomes dos módulos do sistema da empresa;
- Origem da alteração: Cliente, interna ou externa;
- Serviço(s) relacionado(s): Caso a mudança impacte em outros módulos, estes poderão ser adicionados aqui;
- Data da solicitação: Data que o solicitante reportou a mudança.

Para que uma mudança mude seu *status* de PENDENTE para ACEITA, e futuramente para os outros *status*, é realizada a análise da mudança e o analista deve informar mais alguns atributos. Nessa parte, é recomendado o uso do método "7Rs" proposto pela ITIL. Os atributos são:

- Avaliação da mudança: campo descritivo para avaliar a mudança solicitada;
- Partes envolvidas: Setores/Pessoas que estarão envolvidas no processo da mudança;

- Impacto e riscos previstos: ao realizar uma apuração da alteração, caso sejam detectados possíveis riscos ou impactos causados por ela;
- Trajetória de implementação: campo de texto para mencionar o que já foi feito para que a mudança fique pronta;
- Status da Mudança: PENDENTE, ACEITA, REPROVADA, EM IMPLEMENTAÇÃO, EM ESPERA, EM TESTES, DISPONÍVEL;
- Recomendações: campo para inclusão de observações;
- Previsão de entrega: data prevista para a entrega;

Após a implantação da mudança para seu respectivo solicitante, mais dois atributos devem ser informados pela equipe de suporte:

- Sucesso ou insucesso: atributo *boolean* para identificar se a mudança satisfez a necessidade do solicitante.
- Revisão pós-implementação: para informar problemas futuros consequentes da implementação dessa mudança não constatados anteriormente.

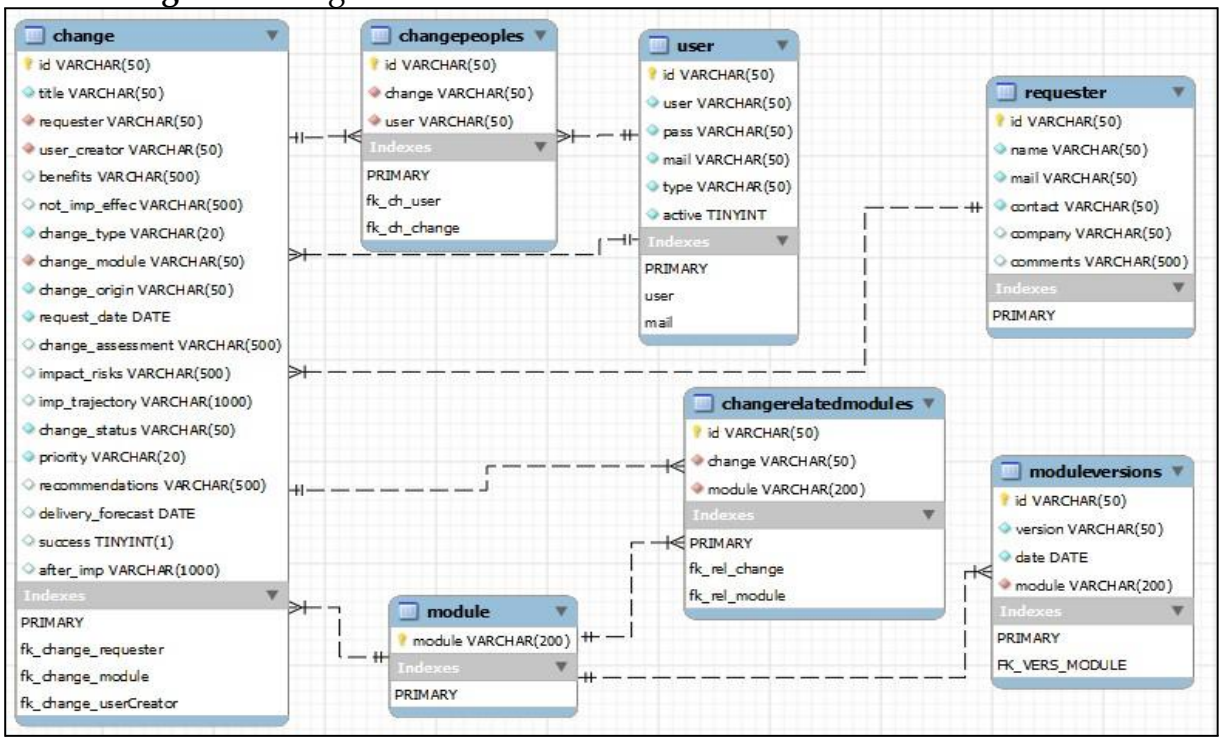
SGMS - Sistema Gerenciador de Mudanças de Software

Para automatizar e apoiar o processo proposto, é desenvolvido um sistema para gerenciamento das mudanças, o SGMS. Os detalhes a respeito do desenvolvimento desse software estão descritos a seguir.

Banco de dados relacional

O diagrama entidade-relacionamento do banco de dados – e, em seguida, a sua criação, utilizando o SGBD MySQL – é apresentado na Figura 3.

Figura 3 - Diagrama Entidade-Relacionamento do Banco de Dados



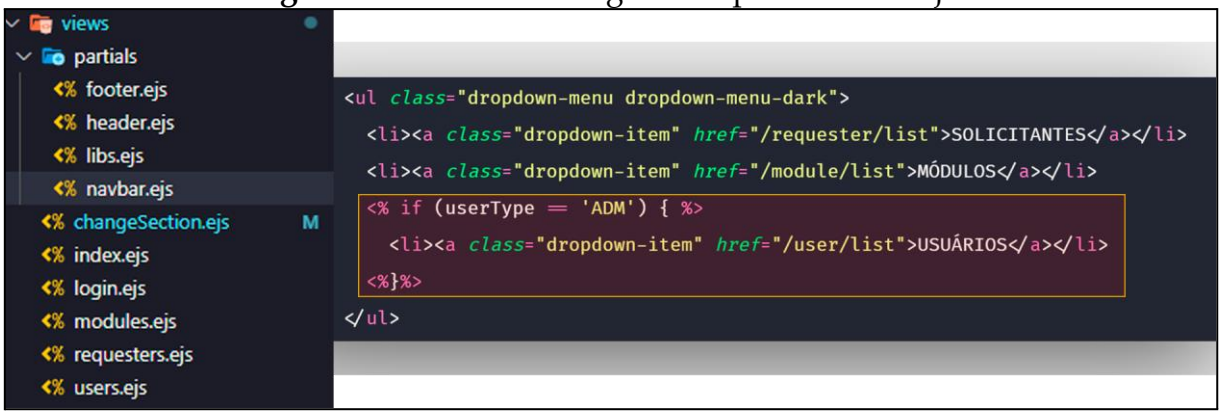
Fonte: elaborada pelos autores.

Codificação do SGMS

Camada View

A camada View (front-end da aplicação) é implementada utilizando o EJS e Bootstrap, sendo a primeira camada de acesso pelo usuário. A Figura 4 mostra a estrutura de arquivos ".ejs" utilizados, sendo a pasta "*partials*" os arquivos que representam partes utilizadas em vários arquivos, além de um trecho de código-fonte destacado usando um condicional no EJS para habilitar ou não o acesso ao cadastro de usuários conforme o tipo de usuário logado.

Figura 4 - Trecho de código do arquivo navbar.ejs



Fonte: elaborada pelos autores.

Camada Controller

A Camada Controller, implementada usando "Express.js", é a camada responsável por receber as requisições da view, instanciar os objetos da camada *model* e realizar a chamada dos métodos de inserção, alteração, exclusão e recuperação de dados. Nessa camada algumas classes tiveram seu arquivo controller individual com seus métodos específicos, como a classe de usuário, por exemplo, que possui métodos de "login" e "logout" implementados em um arquivo controller dedicado à classe, mas os seus métodos CRUD, comuns entre as demais classes, foram implementados no controller CRUDctrl.js.

A Figura 5 mostra um exemplo com o método de inserção e como essa camada implementa a instância dos modelos conforme a requisição recebida.

Figura 5 - Trecho de código-fonte referente aos CRUDs da aplicação

```
function instanciaModel(req) {
  if (req.baseUrl === '/user') {
    model = require('../models/mdlUser');
    return instanciaUser(req.body);
  } else if (req.baseUrl === '/requester') {
    model = require('../models/mdlRequester');
    return instanciaRequester(req.body);
  } else if (req.baseUrl === '/module') {
    model = require('../models/mdlModule');
    return new model(req.body.module, req.body.previous);
  } else if (req.baseUrl === '/version') {
    model = require('../models/mdlVersion');
    return instanciaVersion(req.body);
  } else if (req.baseUrl === '/change') {
    model = require('../models/mdlChange');
    return instanciaChange(req.body);
  }
}
controller.post('/inserir', async (req, res) => {
  try {
    res.status(200).json({ message: await instanciaModel(req).inserir() });
  } catch (error) {
    console.log(error);
    res.status(400).json({ message: `** ERROR!!! **: ${error}` })
  }
})
```

Fonte: elaborada pelos autores

Camada Model

No desenvolvimento da camada modelo do protótipo utilizaram-se alguns conceitos do paradigma de POO, programação orientada a objetos, para o reuso dos métodos de CRUD de todas as classes que os utilizam (change, module, requester, user e version) por herança e polimorfismo (sobreposição).

Nos métodos que realizam requisições ao banco de dados, como “inserir”, “buscar”, “apagar” e “alterar”, foram utilizados os conceitos de requisições bloqueantes. Como os métodos de requisição ao DAO precisam aguardar uma resposta para depois devolver ao controller, foi utilizado o “async” para definir um método assíncrono e o “await” para aguardar a conclusão e resposta do método.

Na Figura 6, pode-se visualizar o modelo de implementação desses métodos na classe pai CRUDModel, sendo o “daoInstance” o objeto de instância da classe DAO. Na Figura 7, as mesmas implementações na classe User, porém, o método de inserir não é implementado, devido à herança.

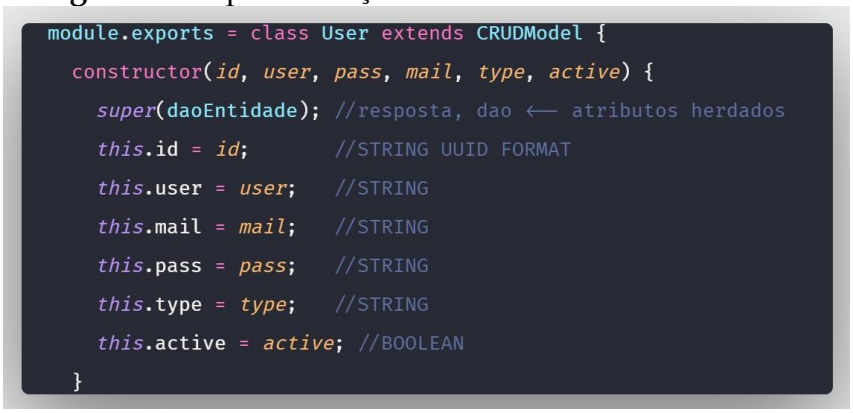
Figura 6 - Implementação da classe pai CRUDModel - Construtor e Inserção

```
module.exports = class CRUDModel {
  constructor(dao) {
    this.dao = dao;
    this.resposta;
  }

  async inserir() {
    try {
      this.validaInserir();
      let daoInstance = new this.dao(this);
      this.resposta = await daoInstance.inserir()
    } catch (error) {
      console.log(error);
      throw error;
    }
    return this.resposta;
  }
```

Fonte: elaborada pelos autores

Figura 7 - Implementação do construtor na classe User



```
module.exports = class User extends CRUDModel {
  constructor(id, user, pass, mail, type, active) {
    super(daoEntidade); //resposta, dao ⟵ atributos herdados
    this.id = id;       //STRING UUID FORMAT
    this.user = user;   //STRING
    this.mail = mail;   //STRING
    this.pass = pass;   //STRING
    this.type = type;   //STRING
    this.active = active; //BOOLEAN
  }
```

Fonte: elaborada pelos autores

Como mostra a Figura 7, a primeira instrução do construtor da classe User é a invocação da classe pai utilizando o método “super”, assim todos os atributos e métodos da classe “CRUDModel” serão herdados.

Para entender o conceito de sobreposição utilizado, as Figuras 8 e 9 mostram os métodos de tratamento que serão executados antes da inserção e implementados na classe pai e na classe filha. Na classe pai, o método apenas exibe uma mensagem no console indicando que não houve tratamento para inserção, enquanto na classe filha ocorrem todos os tratamentos e validações necessárias.

Figura 8 - Métodos de validação de inserção da classe pai

```
validaInserir() { //MÉTODO DA CLASSE PAI
  utils.msgWarning(
    "Validação de INSERÇÃO executada da classe CRUD_GENÉRICA!",
    '\n',
    this
  )
}
```

Fonte: elaborada pelos autores.

Figura 9 - Métodos de validação de inserção da classe filha User

```
validaEntrada() { //MÉTODO DA CLASSE FILHA - USER
  if (!this.user) throw 'Nome de usuário vazio ou Inválido.'
  if (!this.mail) throw 'Email informado vazio.'
  if (!this.pass) throw 'Senha vazia ou inválida.'
  if (!this.type) throw 'Tipo de usuário vazio ou inválido.'
  else this.pass = utils.crypt(this.pass, process.env.SECRET);
}
validaInserir() { //MÉTODO DA CLASSE FILHA - USER
  try {
    this.validaEntrada();
    if (!this.active) this.active = true;
    this.id = uuidv4();
  } catch (error) {
    utils.msgError(error);
    throw error;
  }
}
```

Fonte: elaborada pelos autores.

Camada DAO

A camada DAO foi implementada utilizando os mesmos conceitos da camada *model*. Aqui foi utilizado o *framework* Knex.js, que conseguiu cumprir o papel de acesso ao banco de dados utilizando apenas Js.

Um objeto da camada DAO é instanciado na camada *model*. A cada invocação de método, o objeto é instanciado e contém os dados necessários para realizar as operações. Como o Js não possui o recurso nativo de métodos abstratos, métodos que obrigatoriamente deveriam ser implementados na classe filha, são implementados na classe pai, gerando uma exceção; assim, a falta de sua implementação na classe filha ocasionará um erro na execução dos métodos de CRUD.

A Figura 10 demonstra como foi simulado a implementação de um método abstrato,

"*getParamsInsert()*", assim como o método de inserção da classe pai usando o Knex.js.

Figura 10 - Código-fonte da classe CRUDdao

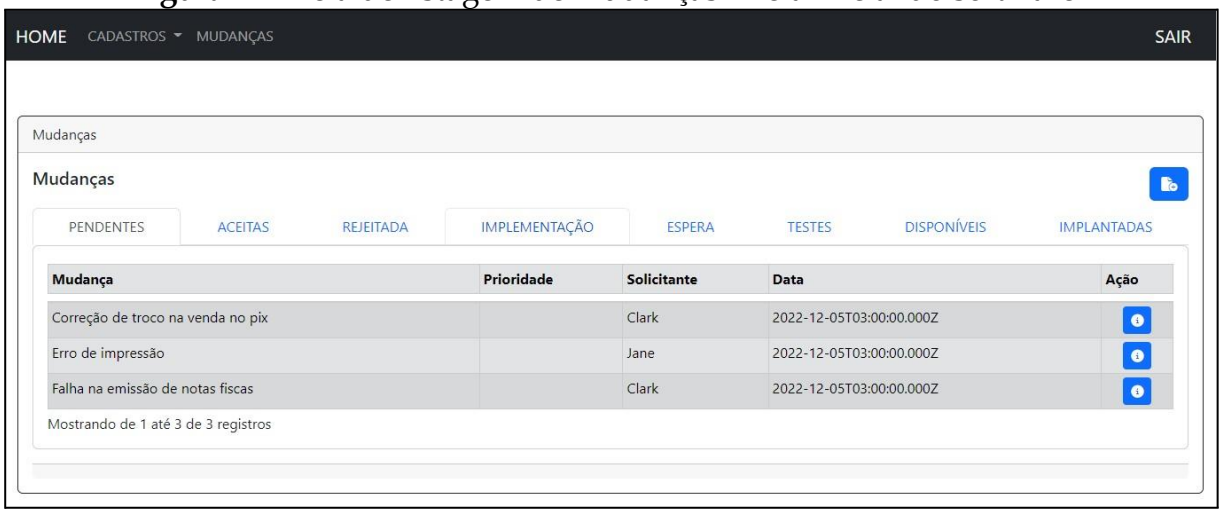
```
getParamsInsert() { //método abstrato para obter os parâmetros para inserção
    msgError('Método não implementado na classe filha.', this.constructor.name)
    throw `Método não implementado na classe filha. ${this.constructor.name}`
  };
  async inserir() { //método de inserção
    try {
      await database //objeto knex representante do banco de dados
        .insert(this.getParamsInsert())
        .into(this.bdTabela)
        .then(() => this.resposta = `'${this.obj.constructor.name}' cadastrado com SUCESSO`)
        .catch((err) => {
          utils.msgError(err);
          throw err;
        });
    } catch (error) {
      utils.msgError(error)
      throw error;
    }
    return this.resposta;
  }
```

Fonte: elaborada pelos autores

RESULTADO

O resultado do desenvolvimento do software é apresentado a seguir. Após o login, é apresentada a tela inicial, que exibe todas as mudanças separadas por *status*. O *status* PENDENTES é a visualização inicial, conforme mostrado na Figura 11.

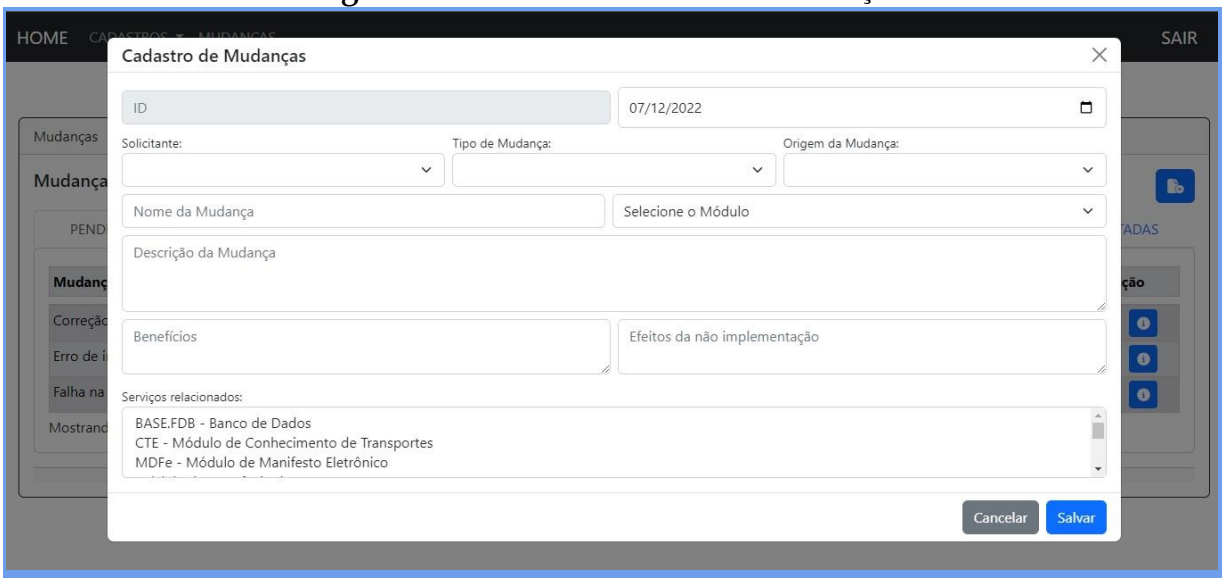
Figura 11 - Tela de listagem de mudanças - Tela inicial do software



Fonte: elaborada pelos autores.

Na tela acima, os usuários do sistema poderão visualizar as mudanças registradas no sistema. Estas estão separadas entre as abas conforme o seu *status*, bastando um clique sobre a aba desejada para listar as mudanças. Aqui os usuários poderão cadastrar novas mudanças clicando no botão de inclusão, que abre um formulário para preencher as informações necessárias, como ilustra a Figura 12. Ainda nessa tela, os usuários poderão clicar sobre o botão de informações da mudança cadastrada para visualizar as informações ou alterar o *status* da mudança, caso este seja um usuário administrador.

Figura 12 - Tela de cadastro de Mudanças

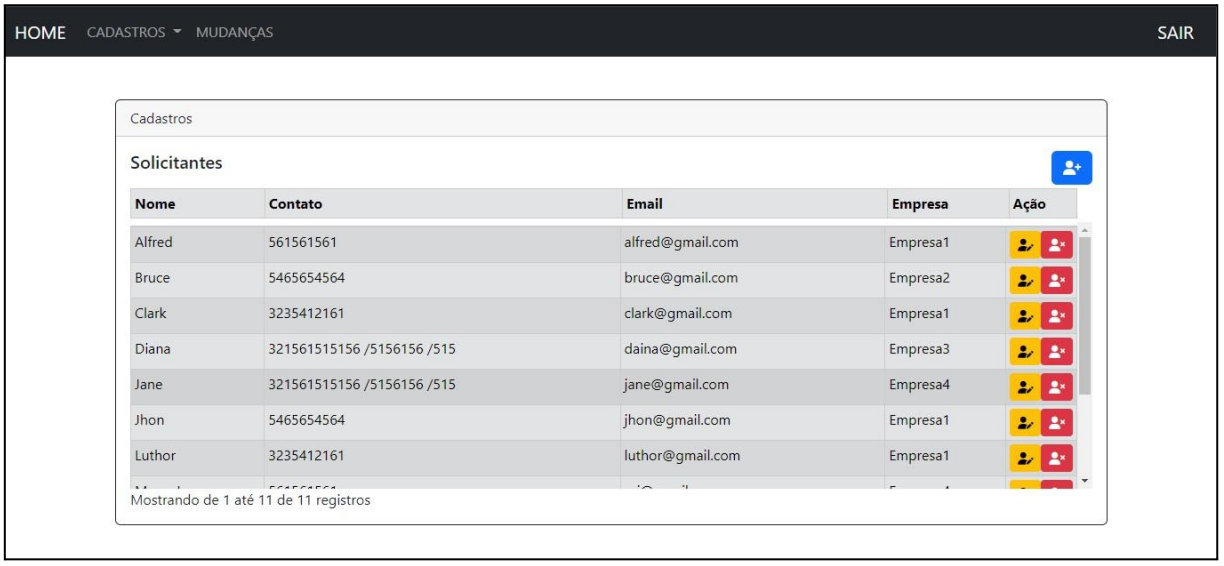


Fonte: elaborada pelos autores.

Para realizar o cadastro de uma mudança, obrigatoriamente o usuário deverá informar o solicitante e o módulo previamente cadastrados. Para realizar os processos de cadastro, visualização e alteração de solicitantes e módulos, o usuário deverá clicar na opção MUDANÇAS e selecionar a opção desejada. Feito isso, o sistema direcionará

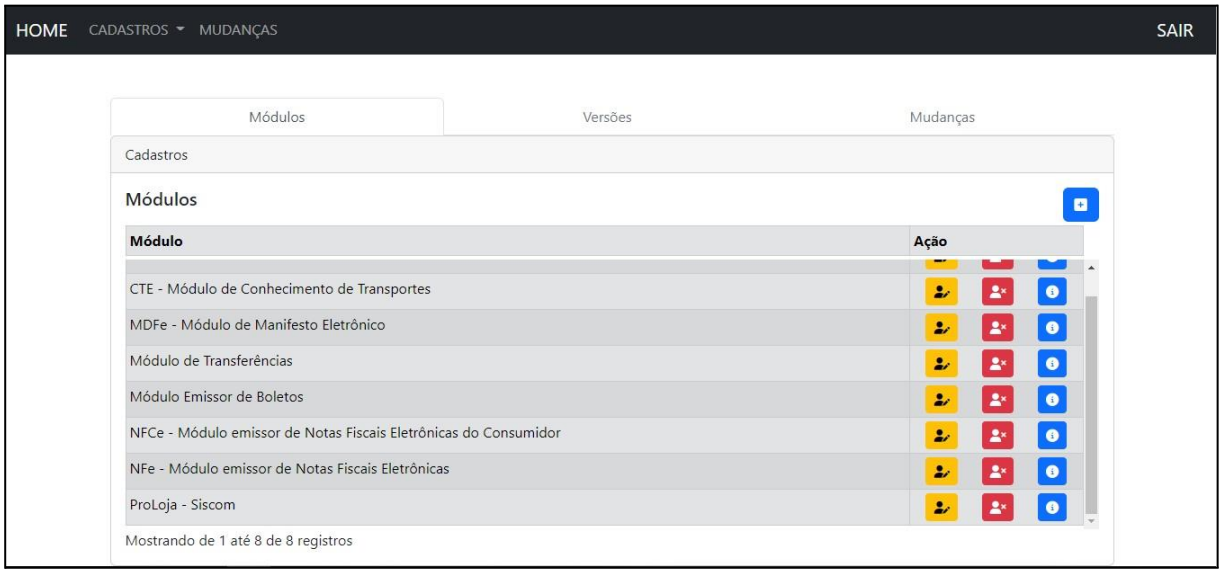
o usuário à sua respectiva página, como mostram as Figuras 13 e 14, para cadastrar os solicitantes e módulos respectivamente.

Figura 13 - Tela de cadastro de Solicitantes



Fonte: elaborada pelos autores.

Figura 14 - Tela de cadastro de Módulos



Fonte: elaborada pelos autores.

CONCLUSÃO

O intuito deste trabalho foi propor uma solução capaz de gerenciar as mudanças solicitadas pelos clientes da empresa Cucabrasil, a qual é embasada nas melhores práticas de gestão de mudanças propostas por autores da engenharia de software e ITIL. Utilizando uma grade de tecnologias modernas, objetivou-se o desenvolvimento de uma ferramenta de simples usabilidade para os usuários, conta com todos os atributos relevantes para uma mudança de software, conforme os conceitos da ITIL, combinados com as necessidades destacadas pela empresa.

O software será apresentado aos integrantes da Cucabrasil e haverá a oportunidade de avaliar e decidir se ele será institucionalizado e utilizado no seu dia a dia. Acredita-se que o processo e a ferramenta proposta podem melhorar o processo de desenvolvimento de software da empresa. Além disso, a ferramenta apresenta grande potencial de evolução, podendo se tornar uma ferramenta capaz de aproximar a empresa ao desejo e necessidades de seus clientes.

Com o intuito de evoluir a ferramenta, pode-se destacar algumas sugestões de trabalhos futuros: a criação de uma interface em que o cliente possa realizar sua solicitação diretamente na ferramenta, a criação de uma interface de acesso para os clientes visualizarem as alterações em produção, sugerir e votar em próximas alterações dando a oportunidade à empresa de conhecer melhor a necessidade do seu público, criar um mecanismo de testes de novas funcionalidades onde o próprio cliente consiga testar, avaliar e reportar problemas antes que essa mudança chegue até ele, implementar filtros nas buscas dos cadastros do sistema.

REFERÊNCIAS

BOOTSTRAP. Crie sites rápidos e responsivos com o Bootstrap. **Bootstrap**. Disponível em: https://getbootstrap.com/. Acesso em: 03 dez. 2022.

DATATABLES. Add advanced interaction controls to your HTML tables the free & easy way. **Datatables**. Disponível em: https://datatables.net/. Acesso em: 03 dez. 2022.

EJS. <%= EJS %> Modelo de JavaScript incorporado. **EJS**. Disponível em: https://ejs.co/. Acesso em: 03 dez. 2022.

FREITAS, Marcos André dos Santos. **Fundamentos do gerenciamento de serviços de TI:** preparatório para a certificação ITIL Foundation. 2. ed. Editora Brasport, 2011.

GOGONI, Ronaldo. O que é software? **Tecnoblog**, 2019. Disponível em: https://tecnoblog.net/responde/o-que-e-software. Acesso em: 27 nov. 2022.

HIGOR. Introdução ao Padrão MVC. **Devmedia**, 2013. Disponível em: https://www.devmedia.com.br/introducao-ao-padrao-mvc/29308. Acesso em: 03 dez. 2022.

KNEXJS. Knex.js Construtor de consultas SQL. **Knexjs**. Disponível em: https://knexjs.org/. Acesso em: 03 dez. 2022.

MOZILLA. Introdução Express/Node. **Mozilla**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs/Introduction#:~:text=O %20Express%20oferece%20solu%C3%A7%C3%B5es%20para,usados%20para%20renderizar%20a%20 resposta. Acesso em: 03 dez. 2022.

MOZILLA. Tecnologia Web para desenvolvedores. **Mozilla**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web. Acesso em: 03 dez. 2022.

MYSQL. Sobre Node.js®. **Mysql**. Disponível em: https://www.mysql.com/why-mysql/. Acesso em: 03 dez. 2022.

NODEJS. Why MySQL? **Nodejs**. Disponível em: https://nodejs.org/pt-br/about/. Acesso em: 03 dez. 2022.

RONALDO. Baixo Acoplamento com DAO. **Devmedia**, 2015. Disponível em: https://www.devmedia.com.br/baixo-acoplamento-com-dao/32696. Acesso em: 03 dez. 2022.

SILVA, Douglas. O que é ITIL e para que serve? Análise detalhada. **Zendesk**, 2021. Disponível em: https://www.zendesk.com.br/blog/o-que-e-itil-e-para-que-serve/. Acesso em: 02 dez. 2022.

SOMMERVILLE, Ian. **Engenharia de software**. 10. ed. São Paulo: Editora Pearson, 2019.