INSTITUTO FEDERAL GOIANO - CAMPUS CERES
BACHARELADO EM SISTEMAS DE INFORMAÇÃO
DANIEL MOREIRA CARDOSO

O USO DO SCRUM NO DESENVOLVIMENTO DE UM SISTEMA DE COMUNICAÇÃO INSTANTÂNEA

CERES - GO
2019

DANIEL MOREIRA CARDOSO

O USO DO SCRUM NO DESENVOLVIMENTO DE UM SISTEMA DE COMUNICAÇÃO INSTANTÂNEA

Trabalho de curso apresentado ao curso de SISTEMAS DE INFORMAÇÃO do Instituto Federal Goiano – Campus Ceres, como requisito parcial para a obtenção do título de bacharel em SISTEMAS DE INFORMAÇÃO, sob orientação do Prof. Me. Adriano Honorato Braga.

CERES - GO

2019

INSTITUTO FEDERAL
Goiano

Repositório Institucional do IF Goiano - RIIF

Goiano

Sistema Integrado de Bibliotecas

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR PRODUÇÕES TÉCNICO-CIENTÍFICAS NO REPOSITÓRIO INSTITUCIONAL DO IF GOIANO

Com base no disposto na Lei Federal nº 9.610/98, AUTORIZO o Instituto Federal de Educação, Ciência e Tecnologia Goiano, a disponibilizar gratuitamente o documento no Repositório Institucional do IF Goiano (RIIF Goiano), sem ressarcimento de direitos autorais, conforme permissão assinada abaixo, em formato digital para fins de leitura, download e impressão, a título de divulgação da produção técnico-científica no IF Goiano.

Identificação da Produção Técnico-Científica

[ ] Tese	[ ] Artigo Científico
[ ] Dissertação	[ ] Capítulo de Livro
[ ] Monografia – Especialização	[ ] Livro
[X] TCC - Graduação	[ ] Trabalho Apresentado em Evento
[ ] Produto Técnico e Educacional - Tipo: _____	

Nome Completo do Autor: Daniel Moreira Cardoso
Matrícula: 2016103202030214
Título do Trabalho: O USO DO SCRUM NO DESENVOLVIMENTO DE UM SISTEMA DE COMUNICAÇÃO INSTANTÂNEA

Restrições de Acesso ao Documento

Documento confidencial: [X] Não [ ] Sim, justifique: _____

Informe a data que poderá ser disponibilizado no RIIF Goiano: 11/12/2019
O documento está sujeito a registro de patente? [ ] Sim [X] Não
O documento pode vir a ser publicado como livro? [ ] Sim [X] Não

DECLARAÇÃO DE DISTRIBUIÇÃO NÃO-EXCLUSIVA

O/A referido/a autor/a declara que:

1. o documento é seu trabalho original, detém os direitos autorais da produção técnico-científica e não infringe os direitos de qualquer outra pessoa ou entidade;
2. obteve autorização de quaisquer materiais inclusos no documento do qual não detém os direitos de autor/a, para conceder ao Instituto Federal de Educação, Ciência e Tecnologia Goiano os direitos requeridos e que este material cujos direitos autorais são de terceiros, estão claramente identificados e reconhecidos no texto ou conteúdo do documento entregue;
3. cumpriu quaisquer obrigações exigidas por contrato ou acordo, caso o documento entregue seja baseado em trabalho financiado ou apoiado por outra instituição que não o Instituto Federal de Educação, Ciência e Tecnologia Goiano.

Ceres, 11/12/2019.

Daniel Moreira Cardoso
Assinatura do Autor e/ou Detentor dos Direitos Autorais

Ciente e de acordo:

Adriano Honorato Braga
Assinatura do(a) orientador(a)

Sistema desenvolvido pelo ICMC/USP
Dados Internacionais de Catalogação na Publicação (CIP)
Sistema Integrado de Bibliotecas - Instituto Federal Goiano

CC268u

Cardoso, Daniel Moreira
O USO DO SCRUM NO DESENVOLVIMENTO DE UM SISTEMA DE COMUNICAÇÃO INSTANTÂNEA / Daniel Moreira Cardoso;orientador Adriano Honorato Braga. -- Ceres, 2019.
72 p.

Monografia (em Sistemas de Informação) -- Instituto Federal Goiano, Campus Ceres, 2019.

1. SCRUM. 2. ENGENHARIA DE SOFTWARE. 3. DESENVOLVIMENTO. 4. METODOLOGIAS ÁGEIS. 5. JAVASCRIPT. I. Honorato Braga, Adriano, orient. II. Título.

Responsável: Johnathan Pereira Alves Diniz - Bibliotecário-Documentalista CRB-1 n°2376

ANEXO IV - ATA DE DEFESA DE TRABALHO DE CURSO

Ao(s) 21 dia(s) do mês de novembro do ano de dois mil e dezenove, realizou-se a defesa de Trabalho de Curso do(a) acadêmico(a) Daniel Moreira Cardoso, do Curso de Bacharelado em Sistemas de Informação matrícula 20161032020302 14, cujo título é "O uso de metodologias ágeis no desenvolvimento de um sistema de comunicação instantânea". A defesa iniciou-se às 13 horas e 05 minutos, finalizando-se às 13 horas e 51 minutos. A banca examinadora considerou o trabalho aprovado com média 7,7 no trabalho escrito, média 8,5 no trabalho oral, apresentando assim média aritmética final 8,1 de **pontos,** estando o(a) estudante apto para fins de conclusão do Trabalho de Curso.

Após atender às considerações da banca e respeitando o prazo disposto em calendário acadêmico, o(a) estudante deverá fazer a submissão da versão corrigida em formato digital (.pdf) no Repositório Institucional do IF Goiano – RIIF, acompanhado do Termo Ciência e Autorização Eletrônico (TCAE), devidamente assinado pelo autor e orientador.

Os integrantes da banca examinadora assinam a presente.

Adriano Honorato Braga

Assinatura Presidente da Banca

Lucas José de Faria

Assinatura Membro 1 Banca Examinadora

Alberto José dos Santos Neto

Assinatura Membro 2 Banca Examinadora

AGRADEÇO A DEUS QUE ME DEU FORÇAS E DISCERNIMENTO, A MINHA FAMÍLIA QUE SEMPRE ME APOIOU E AOS MEUS AMIGOS QUE ME ACOMPANHARAM ATÉ AQUI.

AGRADECIMENTOS

Agradeço primeiramente a Deus pela oportunidade de estudar nessa instituição de ensino federal.

Agradeço também meus pais que sempre me apoiaram nas minhas escolhas e me ajudaram durante toda a jornada do curso.

Ao IF Goiano pelo excelente quadro de docentes que nos ensinaram sempre da melhor forma possível.

Ao meu orientador Adriano Honorato Braga pelo apoio e tempo empenhado.

E por fim a todos os que participaram da minha jornada durante o curso e puderam contribuir com conhecimento, amizade e companheirismo.

Meu sincero muito obrigado!

“A ARTE DESAFIA A TECNOLOGIA, E A TECNOLOGIA INSPIRA A ARTE.”

JOHN LASSETER

RESUMO

Este trabalho descreve o processo de desenvolvimento de um *software* iniciando pela fase do planejamento até a fase do desenvolvimento, isto detalhando a utilização da metodologia ágil, *Scrum*. Além disso, é feito uma revisão bibliográfica das ferramentas atuais no controle de atividades para o desempenhar do processo de desenvolvimento, exemplifica os principais verbos e status do funcionamento do protocolo HTTP. Por fim, retrata as tecnologias utilizadas para o desenvolvimento de uma ferramenta de comunicação instantânea com o intuito de agilizar a comunicação entre os servidores da Empresa X. A utilização de metodologia ágil comparada as clássicas demonstrando a praticidade e dinamicidade do desenvolvimento de *software* que com o *Scrum* se adapta e aperfeiçoa diante da necessidade e especificidade do projeto.

Palavras-chave: SCRUM, ENGENHARIA DE SOFTWARE, DESENVOLVIMENTO, METODOLOGIAS ÁGEIS, JAVASCRIPT.

ABSTRACT

This paper describes the software development process starting from the planning phase to the development phase, detailing the use of the agile methodology, Scrum. In addition, a literature review of current activity control tools to perform the development process is performed, exemplifying the main verbs and status of HTTP protocol operation. Finally, it portrays the technologies used for the development of an instant communication tool in order to speed up the communication between the servers of Empresa X. The use of agile methodology compared to the classic ones demonstrating the practicality and dynamism of the software development that with the Scrum adapts and improves upon the need and specificity of the project.

Keywords: SCRUM, SOFTWARE ENGINEERING, DEVELOPMENT, AGILE METHODOLOGIES, JAVASCRIPT.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BPM	BUSINESS PROCESS MANAGEMENT
REST	REPRESENTATIONAL STATE TRANSFER
HTTP	HYPERTEXT TRANSFER PROTOCOL
WEB	WORLD WIDE WEB
MVC	MODEL VIEW CONTROLLER
UI	USER INTERFACE
ES6	ECMASCRIPT 6
PHP	HYPERTEXT PREPROCESSOR
API	APPLICATION PROGRAMMING INTERFACES
CRUD	CREATE, READ, UPDATE AND DELETE
URL	UNIFORM RESOURCE LOCATOR
JSON	JAVASCRIPT OBJECT NOTATION
XML	EXTENSIBLE MARKUP LANGUAGE
VSC	VERSION CONTROL SYSTEM
SCM	SOURCE CODE MANAGEMENT
CI	CONTINUOUS INTEGRATION
CD	CONTINUOUS DEPLOY
IDE	INTEGRATED DEVELOPMENT ENVIRONMENT
VSCode	VISUAL STUDIO CODE
SGBD	SISTEMA DE GERENCIAMENTO DE BANCO DE DADOS
IP	INTERNET PROTOCOL ADDRESS
ORM	OBJECT RELATIONAL MAPPING

SUMÁRIO

1 INTRODUÇÃO

Comumente, empresas de grande ou pequeno porte, possuem o hábito de se modernizar tecnologicamente, com o intuito de aprimorar, controlar, dar transparência, e na maioria das vezes, automatizar os processos. Ao se deparar com a situação descrita, nos voltamos ao *BPM* (*Business Process Management*), que, segundo (Dirk Krafzig, Karl Banke, 2004), essas iniciativas não abrangem apenas a área de gestão de processos, mas também as voltadas para a qualidade.

Esse tipo de modernização, ainda não está sedimentada por muitas empresas. Seja por motivos de adaptação, por já estarem há algum tempo no mercado, ou pela falta de ferramentas específicas para sua necessidade.

Com o surgimento de metodologias ágeis para desenvolvimento de *software*, principalmente como o *Scrum*, tanto o tempo quanto o custo de desenvolvimento diminuíram. Que, por tais motivos, não informa que o software desenvolvido utiliza essa metodologia seja pior ou menos planejado, pelo contrário, durante as etapas utilizando esse método, percebe-se resultados tão drásticos que atualmente, grandes empresas de desenvolvimento qualificam o método antigo como obsoleto, visto que, são gastos milhões de dólares e nada é entregue e o custo de se manter diagramas coloridos atualizados é muito alto (SUTHERLAND, 2014).

Mudando o modo de desenvolvimento não apenas com conceitos, mas também arquiteturalmente, o modelo *REST* (*Representational State Transfer*), foi adotado como inspiração para a evolução do protocolo *HTTP* (*HyperText Transfer Protocol*), assim como descrito por (FIELDING, 2000), e que hoje, pode ser considerado o modelo de arquitetura mais utilizado na web.

Algumas empresas já familiarizadas no meio tecnológico, algumas vezes acabam perdendo dados de processos por utilizarem meios não específicos de gerenciamento desses processos. Funcionalidades desnecessárias de um sistema genérico para esse fim em específico podem acabar por assustar o meio corporativo, simplificá-lo para a necessidade da empresa e evolui-lo com o tempo e demanda, pode não apenas aumentar a qualidade do gerenciamento dessas informações, mas também produtividade.

O problema se dá a partir de um processo de uma empresa real, que para preservá-la, neste trabalho será denominada como Empresa X, onde os colaboradores das filiais da empresa precisam dar um *feedback* diário sobre as atividades realizadas, que atualmente é feito em um "serviço de mensagens instantâneas baseado na nuvem" (TELEGRAM, 2019), onde não há

a perda dos dados, porém não há o controle ou gerenciamento dos mesmos. A sugestão de solução foi o desenvolvimento de uma aplicação que atenda à essa necessidade específica da empresa.

Portanto, o objetivo deste trabalho é descrever o processo de desenvolvimento de um software, fazendo uso de metodologias ágeis, que agilize o gerenciamento das atividades desenvolvidas dentre os colaboradores da Empresa X.

2 REVISÃO DE LITERATURA

O presente capítulo abordará o referencial teórico do trabalho, sendo a primeira subseção sobre metodologias ágeis aplicadas ao desenvolvimento de *software* em comparação ao modelo em cascata; a segunda subseção abordando arquiteturas de *software* e sua evolução com novas formas de desenvolvimento com a linguagem *JavaScript* e por fim a terceira subseção tratará a respeito do protocolo de comunicação *HTTP*, desde seu início aos principais métodos utilizados.

2.1 METODOLOGIAS DE DESENVOLVIMENTO

O desenvolvimento de *software* tem evoluído constantemente, não apenas com o surgimento de novas tecnologias mantendo o mercado aquecido, mas também na forma de se planejá-lo. Desde o modelo em cascata, que possui seus processos dirigidos a planos e que para cada fase se iniciar, depende da conclusão da fase anterior (SOMMERVILLE, 2003), até métodos ágeis, o planejamento é essencial. Enquanto no modelo em cascata, que o desenvolvimento do *software* somente se inicia ao final da confecção de toda a documentação do software, aprovados e assinados (SOMMERVILLE, 2003), no *Scrum*, a documentação é vista como incremento, se fazendo necessária apenas quando a mesma acrescenta valor (SUTHERLAND, 2014).

Em outras palavras, o modelo cascata incentiva o planejamento de todo o software antes mesmo do início do desenvolvimento, como também a entrega do *software* completo, que diferente do *Scrum*, definidas as histórias ou tarefas, cada *sprint* (termo utilizado no *rugby*, um esporte famoso na Inglaterra, se refere à um ciclo, corrida ou grande avanço), tem por objetivo entregar um módulo ou parte do *software* funcionando (SUTHERLAND, 2014). Por se utilizar dessa abordagem incremental, à cada novo ciclo de planejamento, revisão, entrega e *feedback*, o produto final se aproxima com mais fluidez de acordo com os requisitos dos *stakeholders* (interessado no produto), além de facilitar correções, alterações ou ajustes, caso algum módulo ou funcionalidade não fique como o esperado ou visualizado uma melhor forma de desenvolvê-lo (SOMMERVILLE, 2003).

Assim como no caso da *Medco*, que seu intuito era incentivar a entrega de medicamentos pelos correios e também desenvolver um *software* que seria capaz de gerar relatórios sobre os medicamentos que uma pessoa consome, e cruzar esses dados com os tipos de doenças que a pessoa possui, visto que, geralmente, a maioria dos médicos não trocam informações

entre si sobre pacientes, que apenas as indústrias farmacêuticas possuíam, assim ajudando seus clientes a economizar na compra de medicamentos e também em todos os custos médicos, a equipe responsável por desenvolvê-lo, por utilizar do método em cascata levou seis meses para concluir que não conseguiriam atender ao prazo, então resolveram aplicar o *Scrum* no projeto, convertendo seus incríveis 61 centímetros de documentação em papel, para colunas de tarefas palpáveis e organizadas pela quantidade de trabalho que aquela tarefa geraria, não o quanto iria demorar, mas dificuldade de exercê-la. Com os objetivos traçados, cada equipe sabia exatamente por onde começar, como executar, e quando ela estaria pronta (SUTHERLAND, 2014). Tal metodologia será essencial para este projeto, não apenas para questões de organização, mas como cumprimento de prazos e também entrega de produtos prontos e funcionais em melhor tempo.

Ainda sobre métodos ágeis de desenvolvimento de *software*, vale ressaltar o uso do *Kanban Board* (Quadro de Kanban), junto ao *Scrum*, que por ser uma ferramenta responsável pela administração de produção, disponibiliza visualmente o andamento do processo, controlado pelo responsável do mesmo. Processos que utilizam dessa ferramenta possuem o tempo de espera reduzido e também a produtividade da equipe tende a melhorar (MOURA, 2003).

2.2 ARQUITETURAS DE *SOFTWARE*

As metodologias de desenvolvimento de *software* evoluíram, mas também a forma de desenvolvê-las, mais especificamente em relação às arquiteturas utilizadas em projetos tanto para *WEB* (*World Wide Web* ou Rede Mundial de Computadores) quanto para aplicativos *Desktop* (Ambiente principal do computador).

Os então famigerados *MV* Patterns* (Padrões *MV**), tendo como ator principal o padrão *MVC* (*Model View Controller*), foram fundamentais para essa evolução, determinaram um padrão de projeto que não só era responsável por organizar o projeto como também elevar sua abstração, da então programação estruturada ou procedural para um conceito isolado, dividindo a aplicação em três camadas. *Model*, responsável pela persistência dos dados e em algumas situações pelas validações. *View*, a camada que tem contato direto com o usuário final, o que atualmente é conhecida como *UI* (*User Interface*, Interfaces de usuário) e por fim *Controller*, a camada que gere as requisições dos usuários e determina com qual *Model* se comunicar e qual *View* será exibida. O padrão *MVC* já é bem consolidado, desde seu início em 1979 teve como principais vantagens a simples modularização, manutenção, duplicações do

núcleo de desenvolvimento, agilidade no trabalho em equipe devido a modularidade (OSMANI, 2017).

A partir daí surgem arquiteturas segmentadas tanto no *front-end* (Parte do sistema que cuida das entradas do usuário, as interfaces de usuário), quanto para o *back-end* (O servidor da aplicação), que é responsável por obter as entradas do usuário e gerir os dados.

No desenvolvimento *front-end* a linguagem *ECMAScript*, ou *JavaScript*, como é popularmente conhecida, é "amplamente utilizada para controlar o comportamento de páginas web" (MOZILLA, 2019), que desde 2012, em sua versão 5.1, é compatível com todos os navegadores modernos, já os navegadores antigos possuem compatibilidade com a versão 3. Diversos *frameworks* ou bibliotecas específicas para o *front-end* utilizam a versão 2015 do *JavaScript*, mais conhecida como *ES6* ou *ECMAScript 6*, fazendo necessário o uso de transpiladores, responsáveis pela tradução do código para interpretação do mesmo nos navegadores, mesmo que antigos (MOZILLA, 2019).

A proposta, então, dos *frameworks* ou bibliotecas baseadas em *JavaScript* é a própria *UI*, visto que, ultimamente, muito tem se falado em usabilidade, não apenas na construção de telas bonitas, mas também na facilidade de uso do *software* para o usuário final.

Para concretizar ainda mais a utilização da linguagem *JavaScript*, desde 2015, segundo o (GITHUB, 2019), "uma plataforma de hospedagem de código-fonte com controle de versão usando o Git", ela é a linguagem mais utilizada em projetos, superando o *Java*, e até mesmo o *PHP* (Hypertext Preprocessor). A opinião é a mesma do ponto de vista do (OVERFLOW, 2019), "a maior e mais confiável comunidade online para todos que codificam para aprender, compartilharem seu conhecimento e construir suas carreiras", sendo a linguagem mais pesquisada em 2018.

2.3 PROTOCOLO DE COMUNICAÇÃO *HTTP*

Uma forma de se garantir a comunicação do *front-end* com o *back-end* é a partir do protocolo *HTTP*, através de seus métodos de requisição ou verbos. Um estilo de arquitetura bastante comum em *softwares* desenvolvidos para a *WEB*, é a arquitetura *REST*, obtendo suporte a partir da versão 1.1 do protocolo *HTTP*, e sua existência em 2000 por *Roy Fielding*, que por sua vez também é um dos principais autores do protocolo *HTTP*.

Requisições *REST* permitem a obtenção de informações por meio de *APIs* (Application Programming Interfaces), como também toda e qualquer operação que um *CRUD* (*Create,*

Read, Update and Delete) pode realizar, facilitando a comunicação pelo protocolo *HTTP*, quanto a sua semântica, através de seus verbos (FIELDING, 2000).

Como descrito por *Fielding* em 2000, os verbos do protocolo *HTTP*, semanticamente, são quatro principais: *Get*, o verbo mais utilizado e mais visível, por assim dizer, sendo requisitado sempre que a *URL* (*Uniform Resource Locator*, o endereço virtual) do navegador é alterada; *Post*, o verbo responsável por adicionar informações no servidor; *Put*, responsável por atualizar ou alterar as informações já cadastradas no *software*; *Delete*, como já bem sugestivo pelo nome, é o verbo responsável por apagar as informações.

Requisições *REST* devem possuir um *content-type* (tipo de conteúdo), para especificar o tipo de entrada e retorno, seja ele *JSON* (*JavaScript Object Notation*), o formato mais utilizado, *HTML*, *XML* (*Extensible Markup Language*), entre outras. Considerando o retorno das requisições *REST*, deve-se levar em consideração o *status* (situação), ou códigos de resposta que são enviados juntos ao retorno. Iniciando pelo nível 100 a 199, onde encontramos mensagens de escopo informativo, já os *status* de nível 200 a 299, possuem um escopo de sucesso da requisição, indicando o seu recebimento, aceite e processamento com êxito. Os níveis 300 a 399 indicam redirecionamento de página ou conteúdo, nos níveis 400 a 499 são informados os erros gerados pelo cliente, sendo o status 404 o mais "famoso", indicando que a solicitação não foi encontrada, e por fim os níveis 500 a 599, que são responsáveis por indicar os erros internos do servidor (FIELDING, 2000).

3 DESENVOLVIMENTO

O presente capítulo relata o desenvolvimento do software em si, incluindo a parte de planejamento como também a parte do desenvolvimento e problemas encontrados e soluções para esses problemas.

Serão descritas as histórias, como parte dos requisitos do sistema, avaliando as iterações de todos os atores envolvidos com cada história além de resultados parciais, levando em consideração que cada história gera um resultado referente à história em questão.

Após a análise e descrição dos requisitos, ou histórias, do sistema, serão descritos todas as telas que serão desenvolvidas envolvendo as iterações com os usuários, como parte dos resultados e parte da documentação.

3.1 MATERIAIS E MÉTODOS

Para o levantamento do referencial teórico foi utilizado a busca das seguintes palavras chaves: *Software development*, Metodologias ágeis, *Representational State Transfer*, *Business Process Management*, *JavaScript development*, nas bases de dados *IEE*, *Science Direct*, *Scielo*, *Spell* e por fim em buscas no *Google*. Após a leitura de alguns dos artigos encontrados foi utilizada a técnica bola de neve para expandir a base de artigos e livros.

3.1.1 SCRUM

Uma das partes mais importantes de um projeto se dá pelo seu planejamento inicial, tanto na parte de infraestrutura, arquitetura, como análise dos requisitos até modo de implementação. O Scrum surgiu para solucionar problemas, e também para otimizar as soluções tanto em qualidade quanto em prazos.

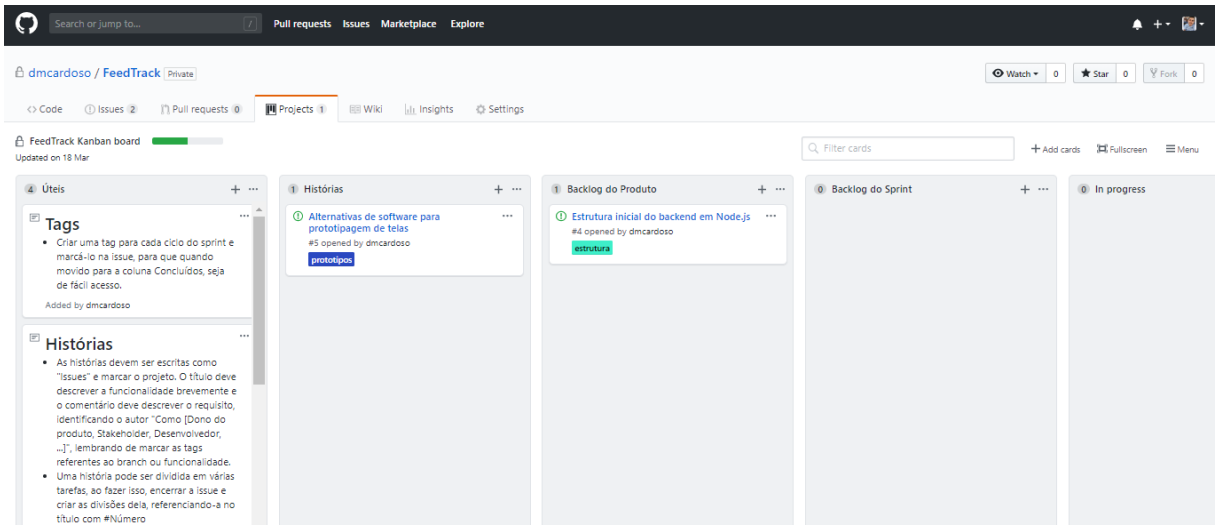
Por utilizar esse *framework*, são agregadas muitas vantagens ao projeto, principalmente na parte da gestão, onde cada história representa uma atividade a ser realizada. Cada história pode ser considerada um requisito para o projeto, seja ele da infraestrutura ou das regras de negócios.

Ao final de todo projeto desenvolvido utilizando o *Scrum*, considera-se como resultado o produto final, em sua maioria em prazo recorde, e também todos os processos utilizados, se bem geridos, considerados a documentação final do produto, visto que as histórias produzidas vão desde a infraestrutura do projeto até os formulários que terão contato direto com o usuário.

3.1.2 *KANBAN BOARD*

Para gerenciamento dos *sprints* e tarefas a serem realizadas no projeto, foi utilizado o gerenciador de projetos disponibilizado gratuitamente pelo GitHub, tendo em vista sua praticidade e ferramentas de automatização das colunas de tarefas com o *issue tracking*, uma ferramenta que, habilitada, é capaz de gerenciar as colunas de Histórias, *Backlog* (Pilha de tarefas) do Produto e Backlog da *Sprint*, automaticamente à partir das *issues* (Problemas do projeto), assim que vinculados ao projeto ou alteração da situação da *issue*, como mostra a Figura 1.

Figura 1 – Interface do gerenciador de Projetos do *GitHub* que utiliza o Quadro de *Kanban*.



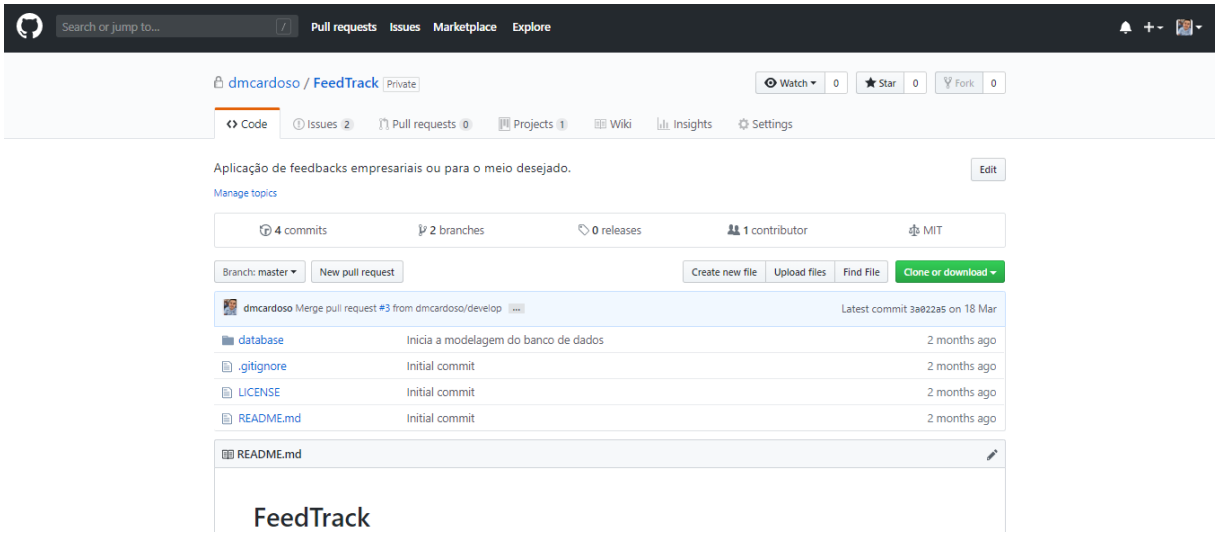
Fonte: Captura de tela do gerenciador de projetos disponível no GitHub em 13 mai. 2019

Avaliadas junto ao *Trello*, e *Pipefy*, o gerenciador de projetos do GitHub foi escolhido pela alta gama de ferramentas, assim como o *Pipefy*, como também o fato de ser gratuito, assim como o *Trello*, mas também pelo fato do GitHub também ser utilizado como *VCS* (*Version Control System*) e *SCM* (*SOURCE CODE MANAGEMENT*).

3.1.3 SISTEMA DE CONTROLE DE VERSÃO - *VCS*

Assim como na parte da gestão do projeto, a ferramenta utilizada para controle de versão e disponiblização da aplicação será o GitHub como interface aplicada ao *Git*, o verdadeiro Sistema de Controle de Versão.

Figura 2 – Interface do gerenciador de repositórios do *GitHub* utilizado como *VCS*.



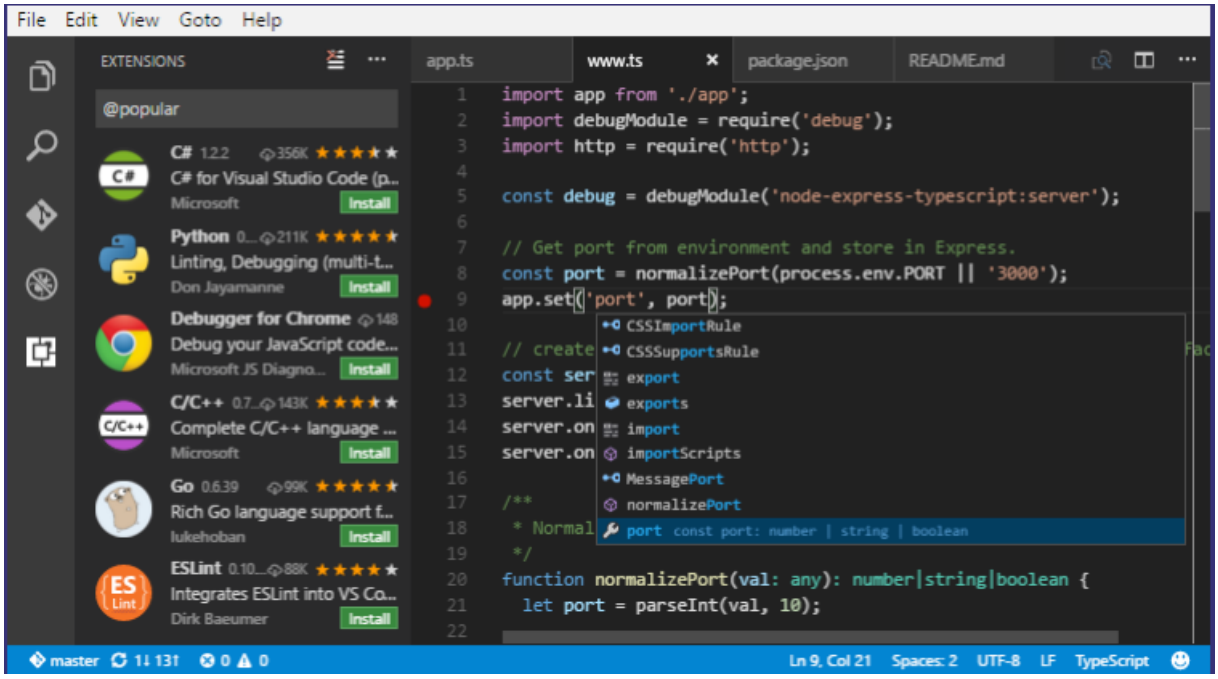
Fonte: Captura de tela do gerenciador de repositórios disponível no GitHub em 13 mai. 2019

Comparado à outros serviços de funcionalidades aplicadas ao *Git*, como o GitLab ou Bitbucket, o *GitHub* sai na frente pela sua interface mais enxuta, como demonstrado na Figura 2, e amigável, além do plano gratuito conter todas as funcionalidades necessárias desde o início do desenvolvimento até a produção e fácil aplicabilidade de práticas como o *CI* (*Continuous Integration* ou Integração Contínua) e *CD* (*Continuous Deploy*, ou Entrega contínua), que são práticas bastante utilizadas para atualização e manutenção de aplicações em produção.

3.1.4 EDITOR DE CÓDIGOS AVANÇADO

Como editor de código, foi utilizado o *Visual Studio Code*, ou VSCode, uma das *IDE's* (*Integrated Development Environment*, ou Ambiente de Desenvolvimento Integrado) construída mais especificamente para desenvolvedores *JavaScript*.

Figura 3 – Interface do editor de códigos avançado, *Visual Studio Code*.



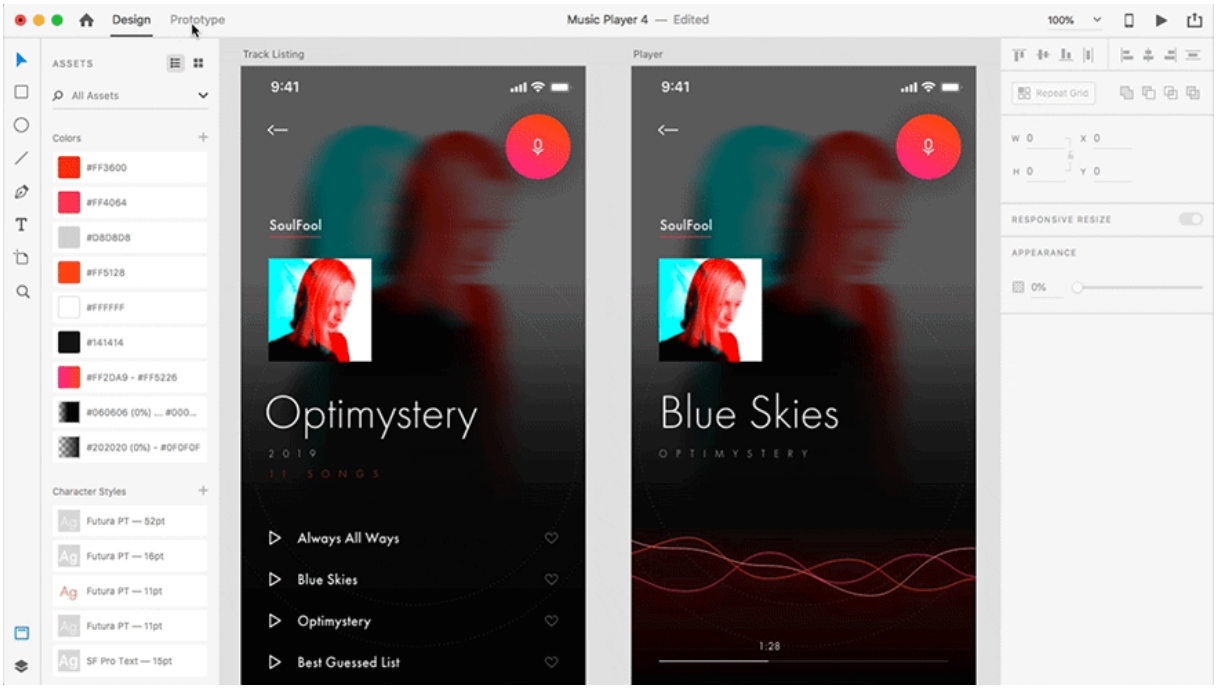
Fonte: Disponível em Visual Studio Code. Acesso em 13 mai. 2019

O *software* é distribuído gratuitamente, *Open Source* (Código Aberto) e mantido pela *Microsoft*. Comparado a outros editores como o *PHP Storm*, da *JetBrains*, ou o *Atom*, desenvolvido pelo próprio *GitHub*, com tecnologias *WEB*, se sobressai em vários quesitos, como a fácil e completa personalização, como demonstrado na Figura 3, através da instalação ou criação de *plugins* (Extensões) personalizadores, além de uma rica gama de extensões que facilitam a codificação, como *IntelliSense* (Autocompletador de código) e até mesmo a formatação e organização do código; e também na velocidade, é um editor muito leve e rápido, diferente do *PHP Storm*.

3.1.5 PROTOTIPAGEM

Foi utilizado como *software* de prototipagem o *AdobeXD*, um software proprietário da *Adobe*, que, até o presente momento, é gratuito, com todas as funcionalidades disponíveis. O *AdobeXD*, é um *software* capaz de criar, de maneira rápida, protótipos e *designs* (Projeto industrial ou projeto de um produto) para interfaces de usuários, por ter uma interface enxuta, assim como na Figura 4, específica para a prototipagem de sites ou aplicativos, torna o aprendizado da ferramenta muito rápido e fácil, e segundo a Adobe em 2019, "foi feito por *designers* e para *designers*".

Figura 4 – Ilustração da interface do *software* de prototipagem *AdobeXD*.



Fonte: Disponível em Adobe. Acesso em 14 mai. 2019

Atualmente, existem vários *softwares* específicos assim como *AdobeXD*, como por exemplo o Figma, uma proposta bastante interessante, por ser um *Web App* (Aplicativo *WEB*), descartando a necessidade de download do mesmo, porém limita algumas funcionalidades caso não adquirida a licença de uso; Sketch, uma outra alternativa de *software* para prototipagem, bastante utilizado para o propósito, porém não há planos gratuitos para sua utilização, e até o presente momento só está disponível para o sistema operacional *macOS*.

3.1.6 INTERFACES DE USUÁRIO

Ainda sobre as interfaces de usuário, ou *UI's*, foi utilizada a biblioteca *ReactJS*, uma biblioteca de propiedade do *Facebook*, *open source*, que sua principal finalidade é a construção de *UI's*, o *front-end* da aplicação, como no próprio sítio do ReactJS diz, "*A JavaScript library for building user interfaces*", ou seja, "Uma biblioteca *JavaScript* para construção de interfaces de usuário"(FACEBOOK, 2019).

Uma biblioteca que trabalha com conceitos de componentização, sejam eles *stateless* (Do inglês, sem estado, ou valor imutável) ou *statefull* (Do inglês, com estado, ou valor mutável), que também podem ser chamados de *Uncontrolled Components* e *Controlled Components*,

respectivamente, e por ser uma biblioteca específica para o *front-end*, a forma de se obter dados, e informações do *back-end*, é através de requisições *REST*, independente da *API*, utilizada.

Ao contrário do que se é visto nas primeiras bibliotecas ou *frameworks JavaScript*, como o *JQuery*, o *ReactJS*, disponibiliza o uso de uma *Virtual DOM* (*Virtual Dynamic Object Manipulation*, ou Manipulação Dinâmica de Objeto Virtual), utilizando uma representação da *DOM* real, pelo JavaScript, fazendo com que o código fique mais limpo, enxuto e remove a necessidade de se utilizar atributos desnecessários nas *tags* (Etiquetas) *HTML* (FACEBOOK, 2019).

Uma das principais vantagens de se utilizar o *ReactJS*, é a existência de uma biblioteca, também disponibilizada pelo *Facebook*, que se chama *create-react-app*. Ela disponibiliza a estrutura do projeto completa, já pronta para o desenvolvimento e produção, com uma *build* (construção), otimizada sem duplicidade das bibliotecas utilizadas, diminuindo o tamanho dos arquivos que serão enviados para a produção. Por ser uma biblioteca, ela permite o uso de qualquer outra biblioteca de terceiros, o que não ocorre no *React Native*, por ser considerado um *framework*, mesmo herdando os fundamentos principais do *ReactJS*.

3.1.7 SERVIDOR DA APLICAÇÃO

Para o *back-end*, ou o servidor da aplicação, foi utilizado o Node.js, "um *runtime JavaScript* desenvolvido com o *Chrome's V8 JavaScript engine*"(NODE.JS, 2019), mais especificamente, um ambiente de execução *JavaScript* para o *back-end* construído com *V8*, um motor *open source*, de alta performance utilizado tanto no *Google Chrome* quanto para o *Node.js*, de propiedade do *Google* (GOOGLE, 2019).

Como principal vantagem de se utilizar *Node.js*, pode-se citar a praticidade de constução de *API's*, com suporte à todos os verbos *HTTP*, com o mínimo de configurações. Para isso, basta instalar e utilizar bibliotecas como o *Express.js*, ou *Koa.js*, bibliotecas que tem como principal objetivo a construção de aplicativos *WEB* e *API's* com alta perfomance e como já citado, configurações mínimas.

Diferente do *PHP*, o *Node.js* não é interpretado, uma vez iniciado, todas as funcionalidades disponíveis no *back-end* estarão em funcionamento, onde só irão ser processadas as funções da funcionalidade requisitada.

3.1.8 BANCO DE DADOS

O *MySQL*, por ser *open source*, gratuito e amplamente utilizado no desenvolvimento *web*, foi escolhido para ser o banco de dados do projeto. Sendo um *SGBD* (Sistema de Gestão de Banco de Dados) com todas essas características, e possuir uma dialética simples e tipagem que atende a todas as necessidades do projeto, não houve necessidade de utilizar outro *SGBD* com mais especifidades.

Classificado como segundo banco de dados mais utilizado, pelo menos até o mês de novembro do ano de 2019, por DB-ENGINES, (2019), o *MySQL* demonstra sua força concorrendo com grandes nomes como *Oracle* e *SQL Server*.

3.2 HISTÓRIAS

A seção de histórias tratará da parte de requisitos, que foram definidos em ordem cronológica e organizados por ordem de relevância no gerenciador de projetos do *GitHub*, gerando mais requisitos conforme a entrega dos requisitos já estabelecidos.

Aproveitando-se da praticidade do *GitHub* a sistematização da organização foi realizada por meio da funcionalidade de automação dos *issues*, onde ao abrir um *issue*, era automaticamente importado para o projeto, para a coluna de *Backlog* do produto, e ao seu término, movido para a coluna de conclusão, seguindo assim, os conceitos o Quadro de *Kanban*.

3.2.1 BANCO DE DADOS INICIAL

A história dessa seção tem a finalidade de descrever o banco de dados inicial da aplicação.

Tabela 1 – História (#1): Banco de dados inicial.

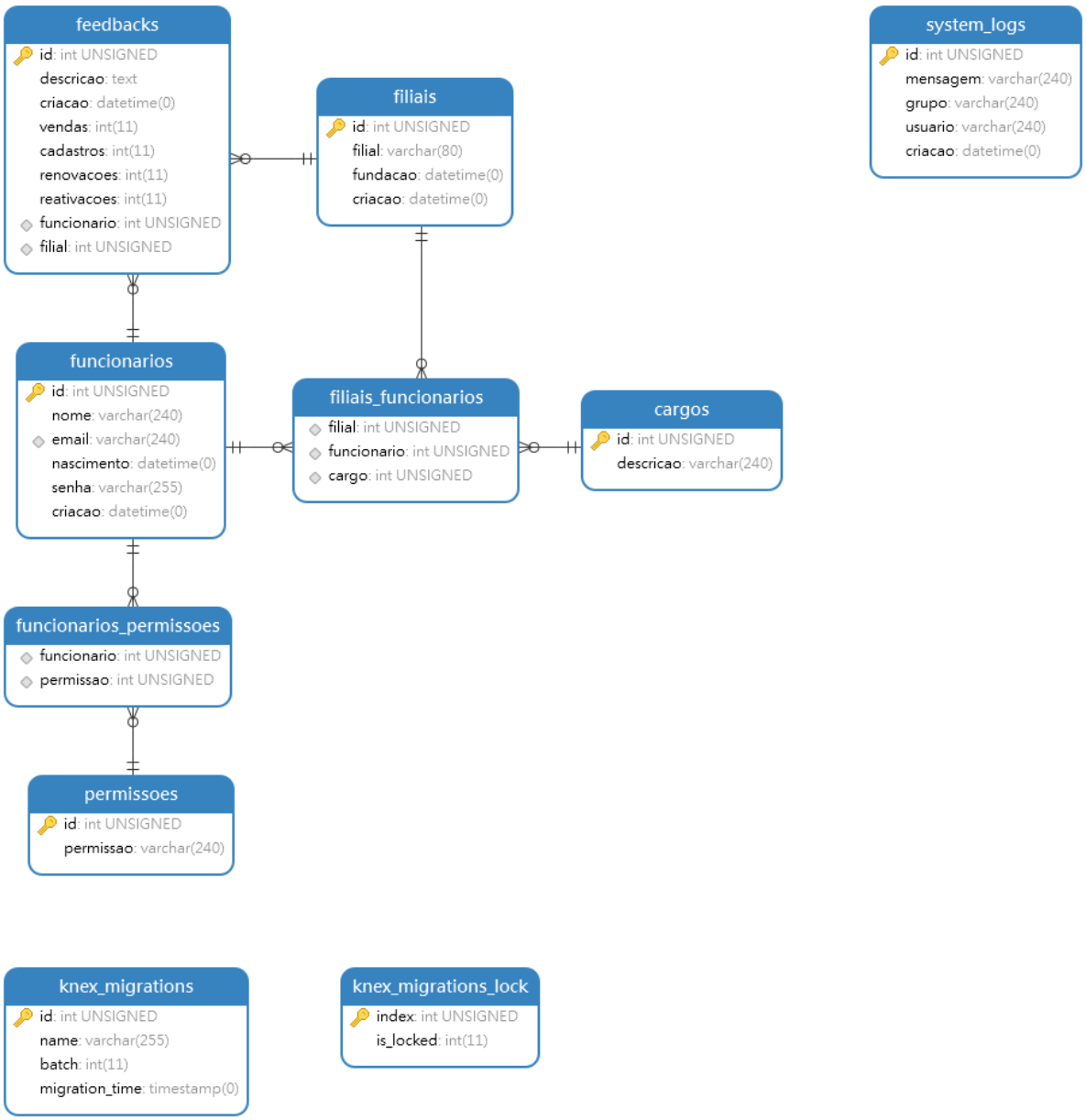
Identificação da história	**#1**
Título da história	Banco de dados inicial
Fonte da história	Análise do sistema
Data	01/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	

Como dono do produto, desejo que seja analisado uma melhor estrutura para o banco de dados, de forma a atender:

- Cadastro de filiais com seus respectivos gerentes, supervisores e funcionários, levando em consideração que um supervisor pode estar em mais de uma filial;
- Cadastro de funcionários com seus respectivos cargos e permissões no sistema;
- Cadastro dos *feedbacks* com data e hora, funcionário, descrição do *feedback*, e nos casos dos supervisores, a filial a qual ele está gerando o *feedback*.

Definida a história, então, é analisado a necessidade a ser atendida, definindo-se então o modelo inicial que será apresentado e descrito posteriormente. Como principais funcionalidades, o sistema deve cadastrar e manter as filiais da empresa, assim como seus funcionários e respectivos *feedbacks*, os quais devem possuir identificadores de horário, dia e caso esteja alocado em mais de uma filial, indicar a filial que possui daquele *feedback*.

Figura 5 – Versão inicial do Modelo Relacional do Banco de dados da aplicação - DER Versão 1.



Fonte: Própria.

Iniciando pelas tabelas *knex_migrations* e *knex_migrations_lock*, representadas na Figura 5, que são responsáveis pela manutenção e atualização entre as versões do banco de dados, são criadas automaticamente pela biblioteca Knex.js, uma excelente ferramenta responsável pela manutenção e atualização de bancos de dados, tanto em ambiente de produção como em ambiente de desenvolvimento.

A tabela de permissões, contendo apenas a descrição da permissão e sua chave primária, como identificador.

A tabela de cargos, é responsável por cadastrar os cargos disponíveis dentro do sistema, contendo a descrição do cargo e sua chave primária como identificador da tabela.

A tabela de filiais, onde serão registradas as filiais cadastradas no sistema, possuindo os campos de filial, onde serão gravados os nomes das filiais, seguidos da data de fundação da filial, e a coluna de criação, onde será registrado a data de quando aquela filial foi cadastrada no sistema.

A tabela de funcionários, onde serão registrados todos os funcionários que poderão manusear o sistema, possuindo campos de informações pessoais como nome, *e-mail* e data de nascimento, e também informações essenciais para o sistema, como senha e data de criação, que se refere à data que aquele funcionário foi cadastrado no sistema.

A tabela *funcionarios_permissoes*, que será uma ligação muitos para muitos, disponibilizando a possibilidade de muitos funcionários possuírem muitas permissões dentro do sistema. Essa tabela será responsável apenas por manter essa relação.

A tabela *filiais_funcionarios*, tem a ideia da tabela descrita anteriormente, onde deverá manter as relações entre a tabela de funcionários e a tabela de cargos, com o acréscimo da tabela de filiais, dando-se a entender que um funcionário pode estar alocado em mais de uma filial com cargos diferentes.

A tabela *system_logs*, que registrará, por meio das regras do software, históricos e registros do sistema, com relações com a tabela de usuário, informando qual usuário realizou a ação em questão, o grupo da ação, que será representado pelo nome da tabela, a mensagem, que será o corpo do registro e a data de criação, que será a data referente à aquela ação em específico.

E por fim, a tabela de *feedbacks*, que será responsável de fato pelos *feedbacks* dos funcionários, possuindo informações como descrição, data de criação, ou a data de quando ele foi cadastrado no sistema, o total de vendas, cadastros, renovações e reativações individualmente, uma relação com o funcionário responsável pelo *feedback* em questão e a filial em que fora realizado.

3.2.2 ESTRUTURA INICIAL DO *BACKEND* EM *NODE.JS*

A história dessa seção tem a finalidade de estruturar a base para o projeto quanto ao lado do servidor, em relação à estrutura de pastas, arquivos, bibliotecas e metodologias que serão utilizadas.

Tabela 2 – História (#2): Estrutura inicial do *backend* em *Node.js*.

Identificação da história	**#2**
Título da história	Estrutura inicial do *backend* em *Node.js*
Fonte da história	Análise do sistema
Data	02/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	
Como dono do produto desejo que seja analisada as melhores bibliotecas, ou *frameworks*, para a definição da estrutura inicial do *backend* do projeto, visando versatilidade, facilidade e não muita complexidade na disposição das pastas referentes aos arquivos visto o tamanho do projeto.	

A linguagem escolhida para lidar com o *backend* da aplicação foi o *Node.js*, por motivos já citados nesse trabalho e pela agilidade e produtividade que ele possibilita na criação de aplicações como é a proposta desse projeto.

Na raiz da pasta referente ao *backend* possuem cinco pastas, das quais uma será responsável por conter todos os arquivos cadastrados no sistema por usuários, uma que é responsável por conter todos os arquivos de criação, alteração e manutenção das tabelas, nomeda *migrations*, outra nomeada de *database*, onde estão localizados os arquivos da gestão da conexão e abstração no manuseio do banco de dados da aplicação. Em seguida a pasta *configs*, que possui arquivos de configuração do sistema, como caminhos para pastas, configurações de *upload* de arquivos, entre outros. E por fim, e mais importante a pasta *app*, que possui o código de todas as entidades e *API's* do sistema.

A pasta raiz também conta com dois arquivos, um nomeado *knexfile.js*, que é responsável pela configuração das conexões dos bancos de dados, tanto para ambiente de desenvolvimento como para ambiente de produção, e o arquivo *index.js*, que é responsável por inicial de fato o servidor.

Código 1 Arquivo responsável por iniciar todo o *backend* da aplicação.

```
1 const app = require('express')();

3 require('./app/middlewares/index')(app);
```

```
4 require('./app/api/index')(app);

6 app.listen(3032, () => {
7     console.log('feedtrack-backend listening on 3032!');
8 });
```

Como já citado, o Código 1 é responsável por iniciar o servidor da aplicação, possuindo a instância da biblioteca *express*, responsável por gerenciar as rotas da aplicação e disponibilizar todo o ambiente para a construção de *API's*, e logo em seguida, a importação dois arquivos muito importantes.

Código 2 Arquivo responsável pela configuração das funções intermediárias de todas as requisições para o *backend*

```
1 const bodyParser = require('body-parser');
2 const cors = require('cors');
3 const express = require('express');
4 const { upload_dir } = require('../../config/paths');

6 module.exports = (app) => {
7     app.use(bodyParser.json());
8     app.use(bodyParser.urlencoded({ extended: false }));
9     app.use(cors());
10    app.use('/images', express.static(upload_dir));
11 };
```

O Código 2 é o arquivo responsável pelas *middlewares*, que são funções que interceptam a requisição e continuam o seu fluxo. Nesse projeto em específico, as middlewares utilizadas são para manipulação dos dados no formato *json*, *URL's* codificadas, *cors*, que habilita a possibilidade de requisições de domínios ou *IP's* (*Internet Protocol Address*, ou Endereço de IP) diferentes, e por último a disponibilidade de arquivos estáticos como imagens, arquivos de texto, entre outros à partir de uma *URL* predefinida, que nesse caso é *images*.

Código 3 Arquivo responsável por carregar e configurar todas as *API's* do *backend*.

```
1 const fs = require('fs');
2 const path = require('path');

4 module.exports = (app) => {
5     fs
6         .readdirSync(__dirname)
7         .filter(file => ((file.indexOf('.')) !== 0    (file !== '
              index.js')))
8         // eslint-disable-next-line import/no-dynamic-require
9         .forEach(file => require(path.resolve(__dirname, file))(app));
```

```
10 };
```

O último arquivo importado, o Código 3, mas não menos importante, é o arquivo responsável por importar os demais arquivos da pasta de *API's* incluindo a instância do servidor em todos eles para que possam ter acesso às outras rotas do sistema e também disponibilizar novas rotas além de conseguir acesso à todas as outras informações disponíveis nesse objeto quanto às configurações, requisições e outras informações.

3.2.3 ALTERNATIVAS DE *SOFTWARE* PARA PROTOTIPAGEM DE TELAS

A presente seção tratará da história cuja finalidade é analisar ferramentas de prototipagem de telas posteriormente ao seu desenvolvimento de fato.

Tabela 3 – História (#3): Alternativas de *software* para prototipagem de telas.

Identificação da história	**#3**
Título da história	Alternativas de *software* para prototipagem de telas
Fonte da história	Análise do sistema
Data	03/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	
Como dono do produto desejo que seja analisado algumas ferramentas com foco em prototipagem de interfaces, com interfaces amigáveis, fácil de manipular e que atenderá às necessidades do projeto.	

Como o intuito dessa história é analisar e escolher a ferramenta que melhor se adéqua aos *designers* e desenvolvedores do projeto, visando a mobilidade entre as plataformas, recursos e conhecimentos já adquiridos, a ferramenta escolhida foi o *AdobeXD*.

O *AdobeXD* é um *software* gratuito, cuja finalidade é a prototipação de interfaces, disponibilizando ferramentas fantásticas para desenvolvedores, como, quando enviado para o modo de desenvolvimento, possibilita a visualização dos estilos de todos os elementos no protótipo, salva exceções de códigos de posicionamento, que requerem um pouco mais de atenção e revisão.

3.2.4 PRÁTICA DE *SOFT DELETE* EM TODOS OS REGISTROS DAS TABELAS

A história dessa seção tem a finalidade de rever os conceitos aplicados ao banco de dados da aplicação, visando alterar a forma como os registros são excluídos evitando, assim, a perca de dados.

Tabela 4 – História (#4): Prática de *soft delete* em todos os registros das tabelas.

Identificação da história	**#4**
Título da história	Prática de *soft delete* em todos os registros das tabelas
Fonte da história	Análise do sistema
Data	04/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	
Como dono do produto desejo que as informações das tabelas não sejam excluídas, apenas desativas, para que não haja perca das informações ou inconsistência dos dados e problemas com as relações de outras tabelas, principalmente a tabela de registros.	

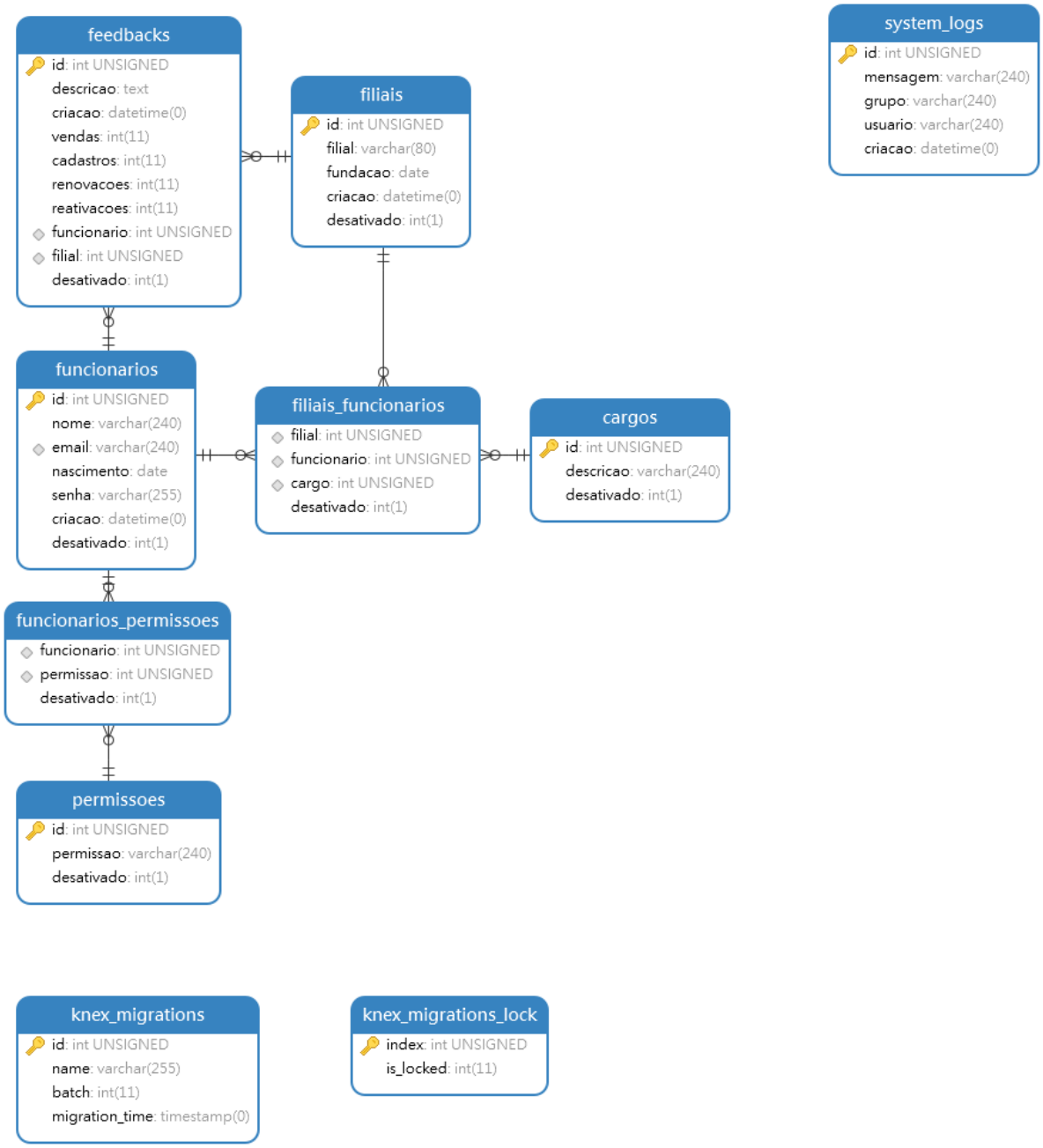
A partir dessa história houve a necessidade de reavaliar o modelo de dados definido anteriormente, visto que era necessário não haver perca de dados.

Após algumas pesquisas com a comunidade contatou-se que a forma mais efetiva de atender tal necessidade é aplicando uma técnica chamada de *soft delete*, cuja finalidade é acrescentar uma coluna no banco de dados nas tabelas em que essa técnica será aplicada, sendo seu valor de apenas um dígito, ou caractere.

Nesse projeto, a técnica de *soft delete* será aplicada em todas as tabelas do sistema, adicionando a coluna desativado cuja lógica será aplicado no sistema validando seu valor por zero, caso haja um valor diferente, o registro será considerado excluído e não visível para qualquer usuário, independente de suas permissões.

A seguir será apresentado o novo modelo relacional após aplicadas as alterações da técnica de *soft delete*.

Figura 6 – Aplicação da técnica de *soft delete* em todas as tabelas do banco de dados - DER Versão 2.



Fonte: Própria.

3.2.5 CONSTRUÇÃO DE *CRUD'S* DAS ENTIDADES

Nessa seção será tratada a construção dos *CRUD's* das entidades do sistema, ou seja, cadastros básicos, manutenção, exclusão e visualização dos dados no sistema, pelo menos das partes principais, foram geradas histórias individuais de todas as entidades que serão apresentadas a seguir.

Tabela 5 – História (#5): *CRUD* - Cargos.

Identificação da história	**#5**
Título da história	*CRUD* - Cargos
Fonte da história	Análise do sistema
Data	05/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	
Como analista do sistema desejo que seja implementado o *CRUD* da entidade de cargos já integrada à *API* ▪ Create; ▪ Update; ▪ Read; ▪ Soft Delete.	

Tabela 6 – História (#6): *CRUD* - *Feedbacks*.

Identificação da história	**#6**
Título da história	*CRUD* - *Feedbacks*
Fonte da história	Análise do sistema
Data	05/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	

Como analista do sistema desejo que seja implementado o *CRUD* da entidade de *feedbacks* já integrada à *API* ▪ Create; ▪ Update; ▪ Read; ▪ Soft Delete.

Tabela 7 – História (#7): *CRUD* - Filiais.

Identificação da história	**#7**
Título da história	*CRUD* - Filiais
Fonte da história	Análise do sistema
Data	05/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	
Como analista do sistema desejo que seja implementado o *CRUD* da entidade de filiais já integrada à *API* ▪ Create; ▪ Update; ▪ Read; ▪ Soft Delete.	

Tabela 8 – História (#8): *CRUD* - Funcionários.

Identificação da história	**#8**
Título da história	*CRUD* - Funcionários
Fonte da história	Análise do sistema
Data	05/07/2019

Responsável pela história	Daniel Moreira
Descrição da história	
Como analista do sistema desejo que seja implementado o *CRUD* da entidade de funcionários já integrada à *API* ▪ Create; ▪ Update; ▪ Read; ▪ Soft Delete.	

Tabela 9 – História (#9): *CRUD* - Permissões.

Identificação da história	**#9**
Título da história	*CRUD* - Permissões
Fonte da história	Análise do sistema
Data	06/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	
Como analista do sistema desejo que seja implementado o *CRUD* da entidade de permissoes já integrada à *API* ▪ Create; ▪ Update; ▪ Read; ▪ Soft Delete.	

Tabela 10 – História (#10): *CRUD* - *System Log*.

Identificação da história	**#10**
Título da história	*CRUD* - *System Log*

Fonte da história	Análise do sistema
Data	05/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	
Como analista do sistema desejo que seja discutido uma melhor forma de otimizar os registros da aplicação.	

Após definidas as histórias, foi iniciada a implementação. Com a utilização da biblioteca *Objection.js*, junto à bibliteca *Knex.js*, a integração da aplicação com o banco de dados é extremamente facilitada, fazendo a escolha do banco de dados a ser utilizado irrelevante.

O *Knex.js* pode ser considerado um *Query Builder* (Fábrica de consultas), e o *Objection*, um *ORM* (*Object Relational Mapping*, ou Mapeador de Objetos Relacionais), que é justamente a proposta de um banco de dados relacional, como o *MySQL*, facilitaram bastante a construção das *API's*.

Foram construídos três métodos principais: *save* (Salvar), *softDelete*, *get* (Pegar, obter). Tais métodos foram implementados com escopo padronizado para todas as entidades, salvas exceções para tratamentos com datas e campos específicos de cada entidade.

Durante o processo de implementação individual das entidades, surgiu a necessidade de automatizar o cadastro de registros, visto que o código responsável por essa inserção era invocado em todas as entidades, o que gerou uma nova história.

Tabela 11 – História (#11): *CRUD - System Logs*.

Identificação da história	**#11**
Título da história	*CRUD - System Logs*
Fonte da história	Análise do sistema
Data	07/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	

Será utilizada a seguinte estrutura:

- id -> Identificador do registro;
- mensagem -> Mensagem customizada para a exibição;
- grupo -> Tabela de referência do registro;
- referencia -> Identificador do registro a ser referenciado;
- usuario -> Usuário que realizou a ação;
- action -> Ação realizada: (*insert*, *update*, *delete*);
- historico -> No caso de ações de *update*, essa coluna é preenchida com os valores alterados de cada campo;
- criacao -> Data em que a ação foi realizada.

Nos casos das tabelas com relacões n n, será cadastrado os ids referentes à relação por meio de caracteres especiais na mensagem para posteriormente serem interpretados e exibidos.
Ex.: Usuário {{usuario}} {{action}} as permissões {{p_id_1}}, {{p_id_2}} e {{p_id_3}}, do usuário {{u_id_6}}.
Usuário Pedro removeu as permissões Concluir chamado, Cadastrar Filial e Cadastrar Cargos para o usuário Augusto.

Para aplicar a estratégia definida na história anterior, foram desenvolvidos dois algoritmos genéricos na classe abstrata que faz parte dos arquivos de configuração do banco de dados.

Código 4 Algoritmo responsável pela inclusão dos registros do sistema na tabela de histórico dos registros do sistema.

```
1 .runAfter(async (result) => {
2   const log = {
3     // eslint-disable-next-line max-len
4     mensagem: `${model_class.labelName} {{${model_class.tableName}_id
          }} ${action} pelo usuario {{usuario_id}}`,
5     grupo: model_class.tableName,
6     usuario: model.user.id,
7     referencia: model.id,
8     action,
9     historico: getModelDiff(model, modelBefore),
10  };
```

```
12    if (result) {
13      try {
14        // eslint-disable-next-line import/no-dynamic-require
15        const SystemLogs = require(path.resolve(model_class.modelPaths,
              'system-logs.js'));

17        SystemLogs.query().insert(log).then();
18      } catch (msg) {
19        throw msg;
20      }
21    }

23    return result;
24  });
```

O algoritmo representado no Código 4 é executado sempre após qualquer uma das ações genéricas do sistema, seja ela métodos de inserção, atualização ou exclusão. Ela é responsável por inserir ta tabela de registros as ações, mensagens e histórico dos registros de todas as entidades do sistema.

Código 5 Algoritmo responsável por identificar as alterações dos dados a serem alterados.

```
1  const getModelDiff = (model, modelBefore) => {
2      if (!modelBefore) {
3          return null;
4      }

6      let diffMessage = 'Alteracoes\: \\n';
7      let diffFieldsCount = 0;
8      const diffs = [];

10     const messageTemplate = (field, oldValue, newValue) => `${field}
           alterado de '${oldValue}' para '${newValue}'`;

12     Object.keys(model).forEach((field, index) => {
13         const oldValue = modelBefore[field];
14         const newValue = model[field];

16         if (oldValue !== newValue) {
17             diffs.push(messageTemplate(field, oldValue, newValue));
18             diffFieldsCount += 1;
19         }
20     });

22     if (diffFieldsCount > 0) {
23         diffMessage += diffs.join(',\n ');
24         return diffMessage;
25     }

27     return null;
```

```
28 };
```

O algoritmo presente no código 5 teve como intuito a manutenção das informações alteradas no uso do sistema, identificando os valores alterados e retornando-os para o algoritmo anterior complementando assim o registro de *logs* do sistema.

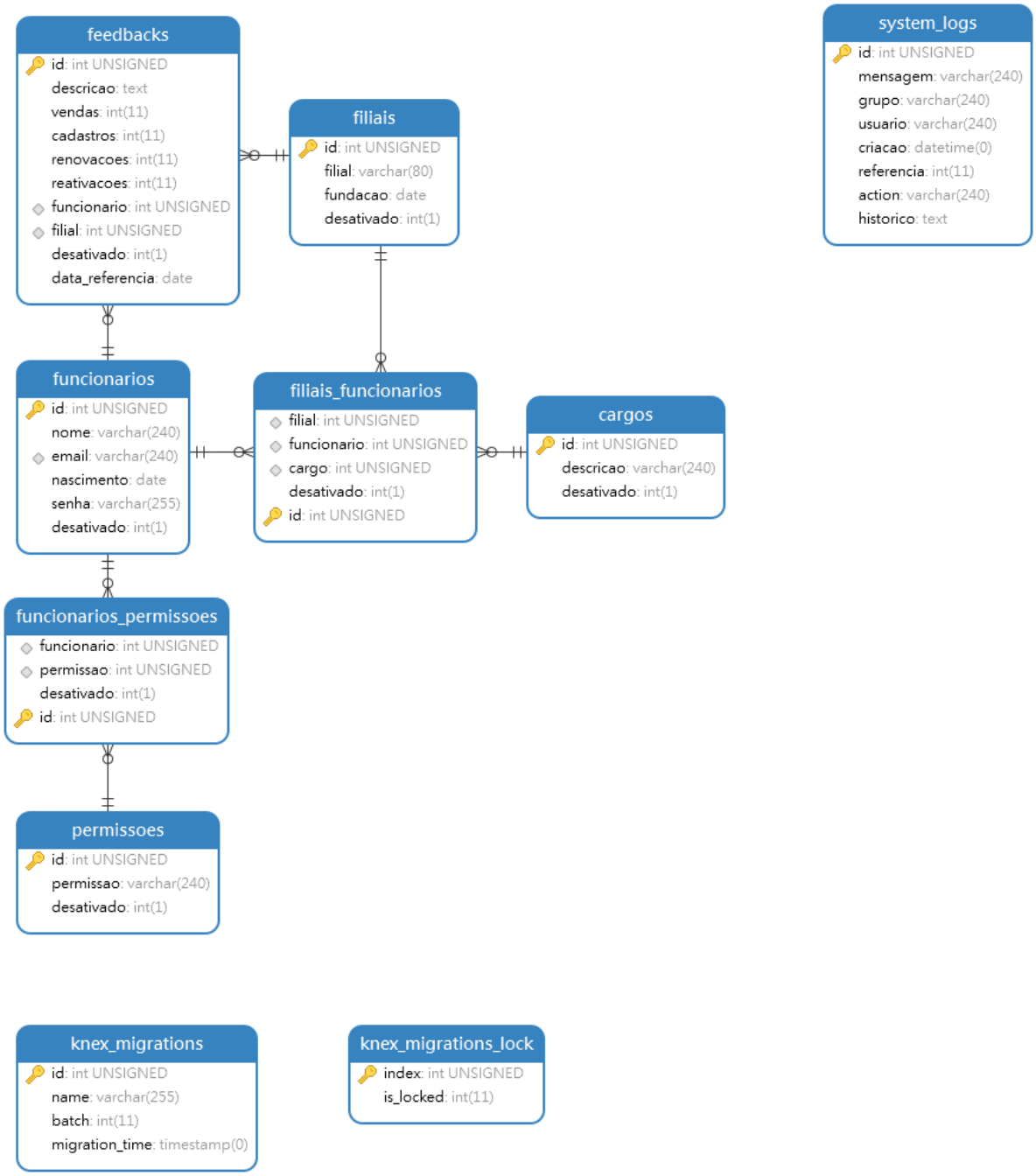
Após essas alterações notou-se que uma modificação na tabela de registros seria necessária e que não se faz mais necessária a coluna que registra a data de criação do registro nas tabelas, gerando então uma nova história.

Tabela 12 – História (#12): Exclusão de todas as colunas de criação nas tabelas do sistema.

Identificação da história	**#12**
Título da história	Exclusão de todas as colunas de criação nas tabelas do sistema
Fonte da história	Análise do sistema
Data	08/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	
Como analista do sistema desejo que sejam excluídas as colunas de criação responsáveis por registrar a data de criação daquela informação, visto a automação do mesmo através da tabela de registros descrita na história 10.	

Após a exclusão e alteração nas tabelas tanto de registros do sistema quanto nas que possuíam a coluna criação, gerou-se um novo modelo relacional.

Figura 7 – Modelo Relacional atualizado aplicando as alterações das histórias da seção de *CRUD's* - DER Versão 3.



Fonte: Própria.

3.3 TELAS

A presente seção tratará a respeito das características de todos os elementos contidos nas interfaces que terão contato direto com o usuário, que, nesse caso específico serão os colaboradores da empresa que utilizará o sistema.

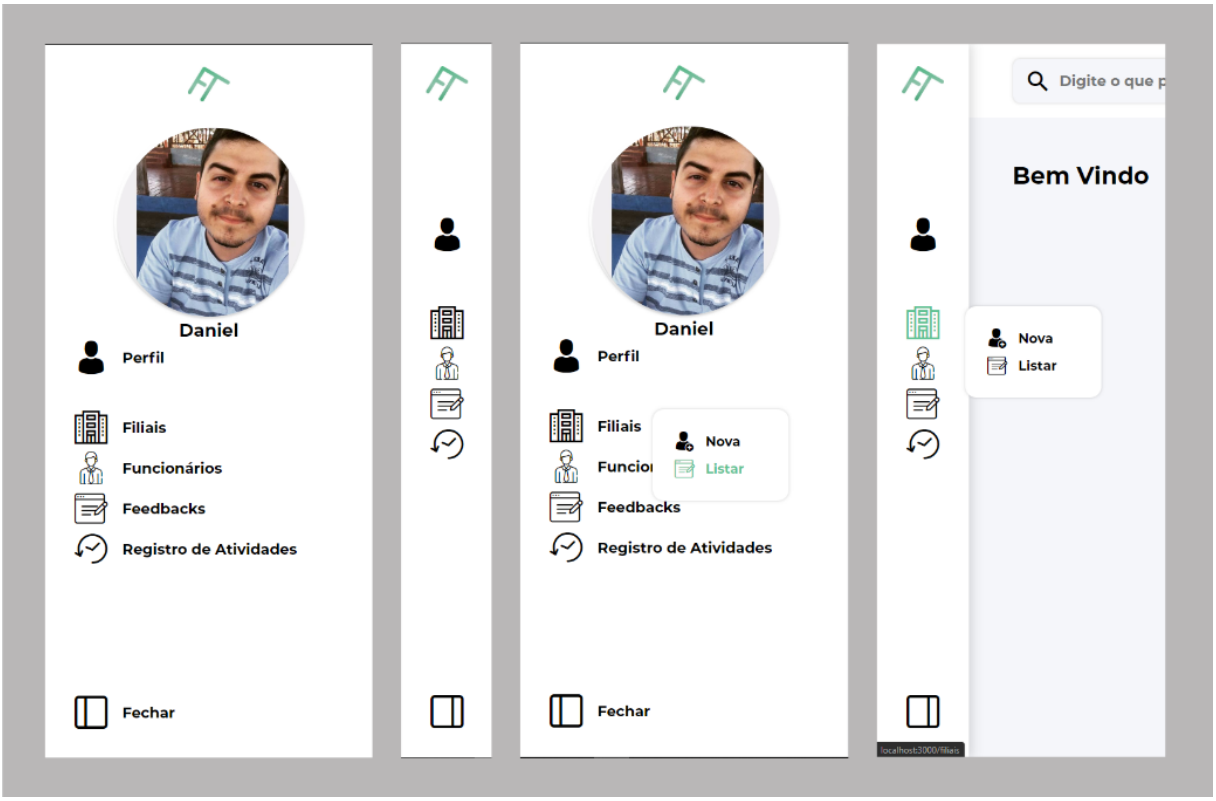
Por ter sido desenvolvido com *React.js*, o termo componente será amplamente utilizado, visto que pelos conceitos da biblioteca, cada interface ou parte dela é considerado um componente.

A dinâmica dessa seção anterior será continuada, será apresentado uma história que gerou as características e individualidades de cada elemento e tela do projeto seguido de uma imagem como resultado daquela história, que, pelo intuito dessa seção será uma página ou um componente que estará presente na tela.

3.3.1 COMPONENTES PRINCIPAIS DO SISTEMA

Dentre os componentes ou telas que compõem o sistema, está a barra lateral, responsável pela navegação no sistema, onde se localizam os botões de ação, seja navegar entre as páginas, tanto de listagem, como de cadastros, como o botão que minimiza a barra lateral, além da presença da logo do sistema e a foto do usuário que está operando o sistema.

Figura 8 – Ilustração do componente principal responsável pela navegação entre as páginas do sistema.



Fonte: Própria.

O botão "Perfil" direciona o usuário para suas informações pessoais cadastradas no sistema, com possibilidade de serem alteradas, caso tenha permissão para tal.

O botão "Filiais" possui um menu flutuante com redirecionamentos para a tela de listagem ou cadastro de filiais, e caso seja clicado no mesmo será redirecionado automaticamente para a tela de listagem de filiais.

O botão "*Feedbacks*" também possui um menu flutuante com os mesmos redirecionamentos que o botão de filiais, sendo possível visualizar, editar, cadastrar ou excluir *feedbacks* existentes.

O botão "Registro de Atividades" diferente dos outros, não possui um botão flutuante, visto que o registro de atividades serão os registros do sistema, de criação, atualização e exclusão, então só há a opção de listagem, que é acionada ao clicar no botão.

E por fim, o botão "Fechar", que é responsável por diminuir, e após pressionado, aumentar, a barra lateral, assim como demonstrado na Figura 8.

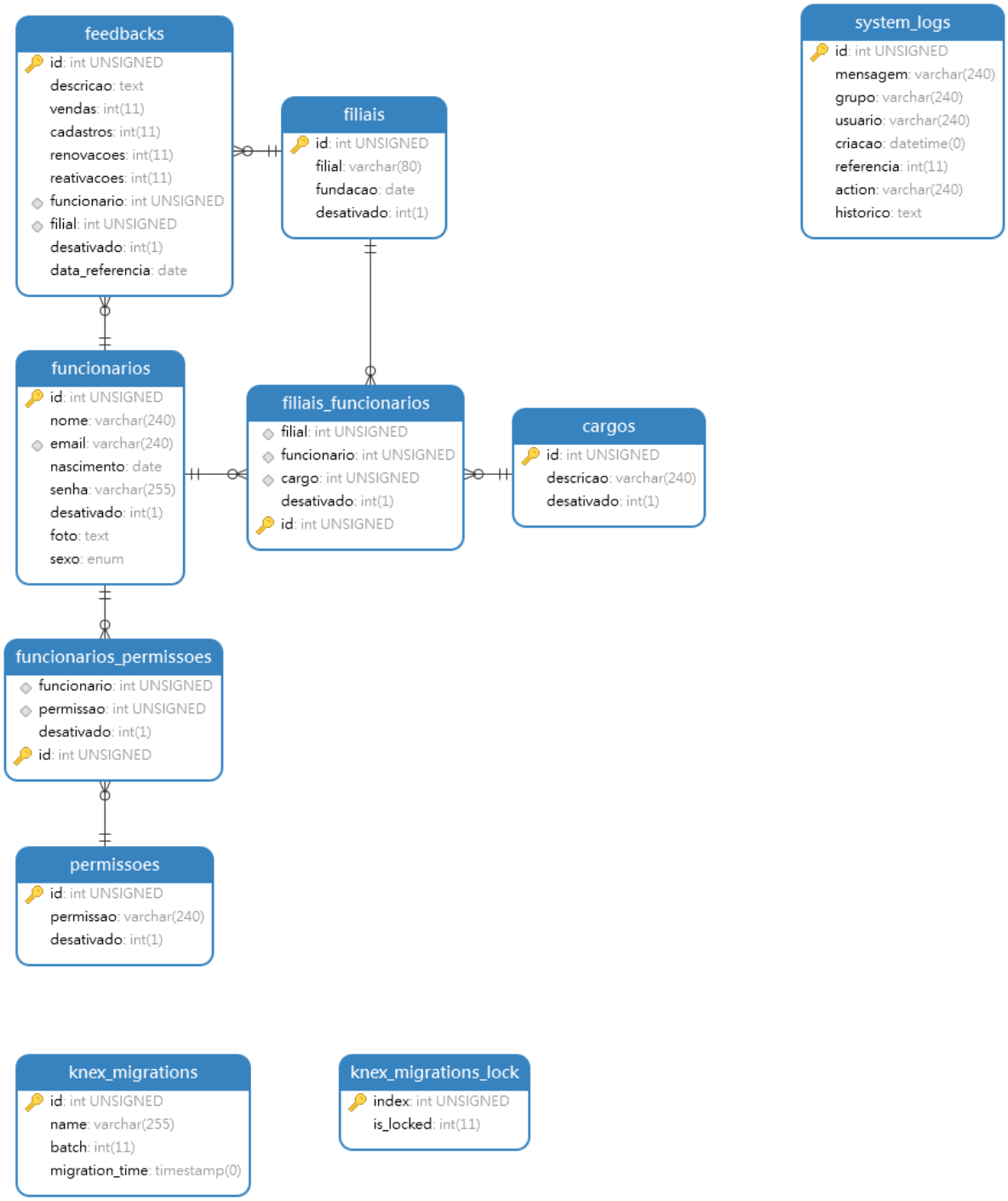
Após a prototipação e desenvolvimento da tela viu-se a necessidade de cadastrar no banco de dados a foto do usuário, para adaptação e complementação da interface, gerando assim, um nova história para ser desenvolvida.

Tabela 13 – História (#13): Suporte a foto e sexo no cadastro de funcionários.

Identificação da história	**#13**
Título da história	Suporte a foto e sexo no cadastro de funcionários
Fonte da história	Análise do sistema
Data	09/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	
Como dono do produto desejo que seja atualizada as informações registradas para os usuários cadastrados no sistema, como foto e sexo, a fim de aprimorar e adaptar a interface do sistema.	

Devido a necessidade da história modificar a estrutura do banco de dados, também foi gerado um novo modelo relacional.

Figura 9 – Modelo relacional do banco de dados com alterações de sexo e foto na tabela de funcionários - DER Versão 4.



Fonte: Própria.

3.3.2 EXIBIÇÃO DE INFORMAÇÕES

As páginas de listagem de informações foram projetadas a partir de dois componentes principais, o título de referência da página, que informa ao usuário a páginas em que ele está,

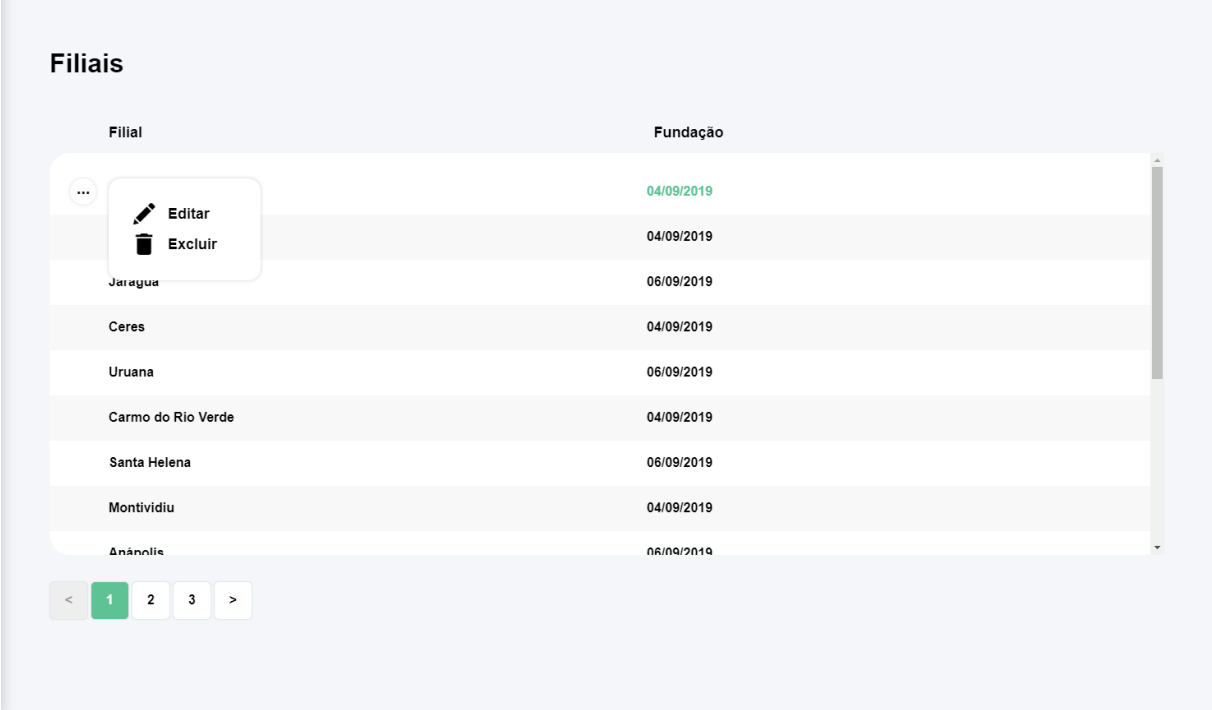
e a tabela responsável pela listagem de dados, possuindo três listagens principais, a listagem de filiais, funcionários, *feedbacks* e registro de atividades, sendo as informações exibidas nessa listagem requiridas através de suas respectivas histórias, tendo como base a história que determinava um padrão para as páginas de listagem.

Tabela 14 – História (#14): Padrão de exibição das informações.

Identificação da história	**#14**
Título da história	Padrão de exibição das informações
Fonte da história	Análise do sistema
Data	10/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	
Como desenvolvedor desejo que seja prototipado um padrão para as páginas responsáveis pela exibição das informações, a fim de padronizar as interfaces e componentes principais dessas páginas.	

Após finalizado os protótipos das telas, foram desenvolvidos os componentes de tabela, representados na Figura 10, com seu submenu dinâmico e o componente de paginação, responsável por navegar na tabela entre os dados registrados no sistema.

Figura 10 – Ilustração dos componentes responsáveis pela listagem das informações.



Fonte: Própria.

As ações padrões do menu são as opções de edição, que ao clicado, o usuário é direcionado para a tela de edição da linha da tabela em que foi clicado e o botão de exclusão, que ao clicado é exibido uma janela flutuante confirmando a exclusão do registro em que foi clicado. Todas as linhas da tabela ao serem clicado também são direcionados à página de edição do registro que foi clicado.

A página que exibirá as informações das filiais cadastradas no sistema, a fim de propor a navegação entre as páginas de edição e cadastro das mesmas.

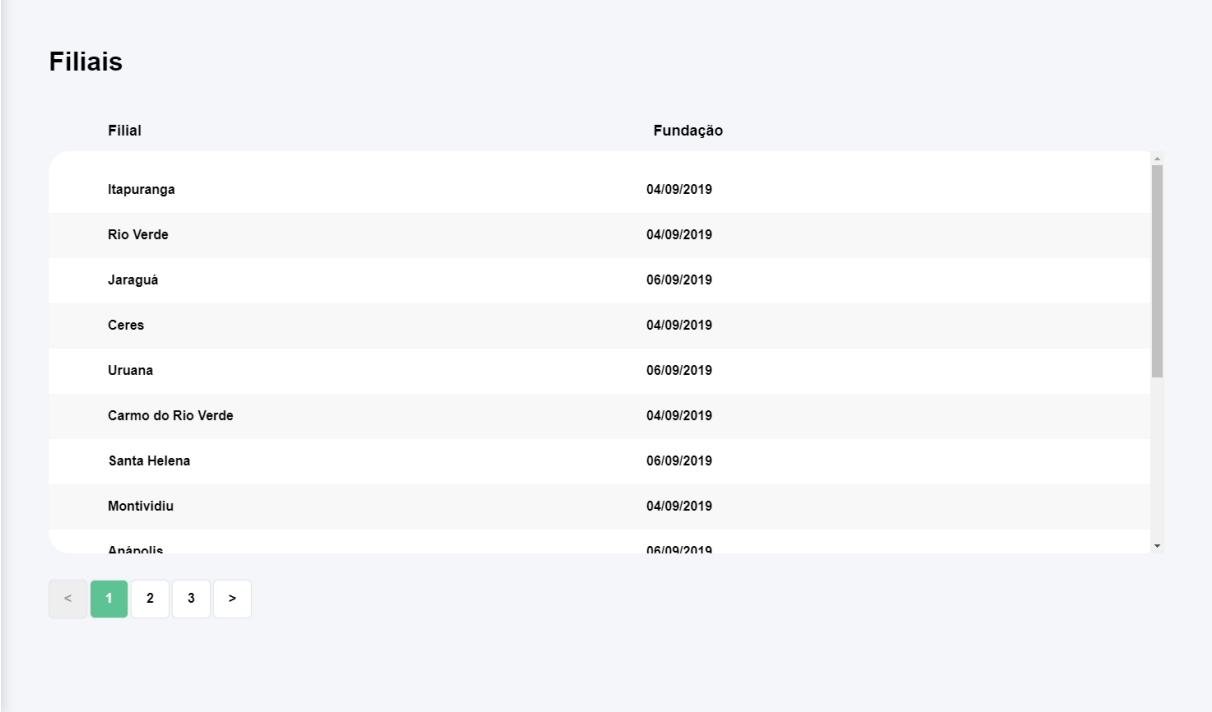
Tabela 15 – História (#15): Listagem de filiais.

Identificação da história	**#15**
Título da história	Listagem de filiais
Fonte da história	Análise do sistema
Data	10/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	

Como dono do produto desejo que seja implementada a página de listagem das filiais cadastradas no sistema, exibindo seu nome e a data de fundação da mesma caso possua.

A princípio, a página de listagem, na Figura 11, exibirá apenas as informações principais que seu cadastro possui, sendo possível alterá-la caso haja necessidade de exibição de outras informações na página de listagem.

Figura 11 – Página de listagem de filiais.



Fonte: Própria.

Na página de listagem de funcionários serão exibidos os funcionários cadastrados no sistema, propondo a navegação para suas páginas de cadastro ou edição e também possibilitando sua exclusão.

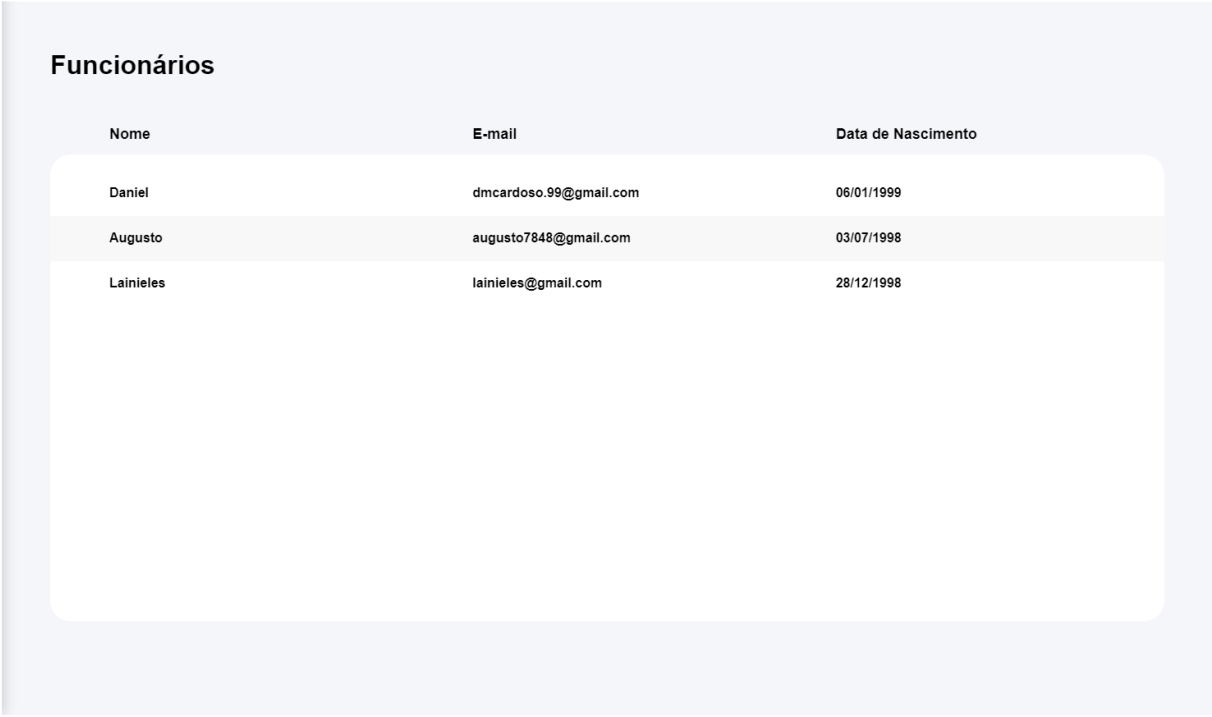
Tabela 16 – História (#16): Listagem de funcionários.

Identificação da história	**#16**
Título da história	Listagem de funcionários
Fonte da história	Análise do sistema

Data	11/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	
Como dono do produto desejo que seja implementada a página de listagem dos funcionários cadastrados no sistema, exibindo informações como nome, *e-mail* e data de nascimento.	

As informações exibidas na página de listagem dos funcionários foram limitadas ao nome, *e-mail*, e data de nascimento, assim como na Figura 12, visto que a primeiro momento não seriam necessárias outras informações, levando em consideração que um funcionário pode preencher mais de uma função em uma ou mais filiais.

Figura 12 – Ilustração da página responsável pela listagem dos funcionários.



Fonte: Própria.

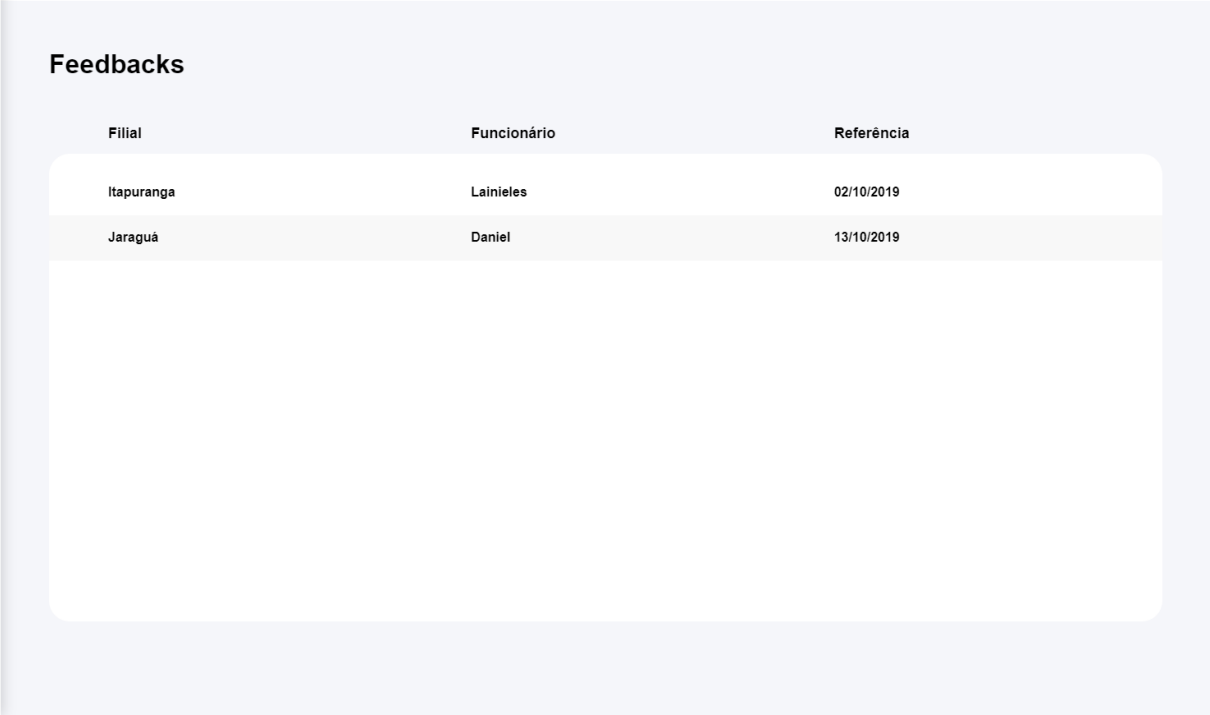
A página de listagem de *feedbacks* exibirá todos os *feedbacks* cadastrados no sistema, propondo a navegação entre as páginas de edição e cadastro no sistema e também possibilitando sua exclusão.

Tabela 17 – História (#17): Listagem de *feedbacks*.

Identificação da história	**#17**
Título da história	Listagem de *feedbacks*
Fonte da história	Análise do sistema
Data	11/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	
Como dono do produto desejo que seja implementada a página de listagem dos *feedbacks* cadastrados no sistema, exibindo informações como a filial retentora do *feedback* em questão, o funcionário responsável por aquele *feedback* e a data de referência do mesmo.	

As informações exibidas na página de listagem dos *feedbacks* requiridos pela história anterior, foram a filial, funcionário e data de referência, como demonstrado na Figura 13, não se fazendo necessário, pelo menos a primeiro momento, a exibição de outras informações como a quantidade das vendas, renovações, cadastros, reativações ou descrição realizadas naquele dia em questão, e caso seja necessário visualizar essas informações basta navegar para a tela individual do *feedback*.

Figura 13 – Ilustração da página de listagem dos *feedbacks*.



Fonte: Própria.

A página de listagem dos registros de atividades exibirá todos os registros do sistema, possibilitando a visualização dos mesmos através de uma janela flutuante ao clicar nos registros da tabela.

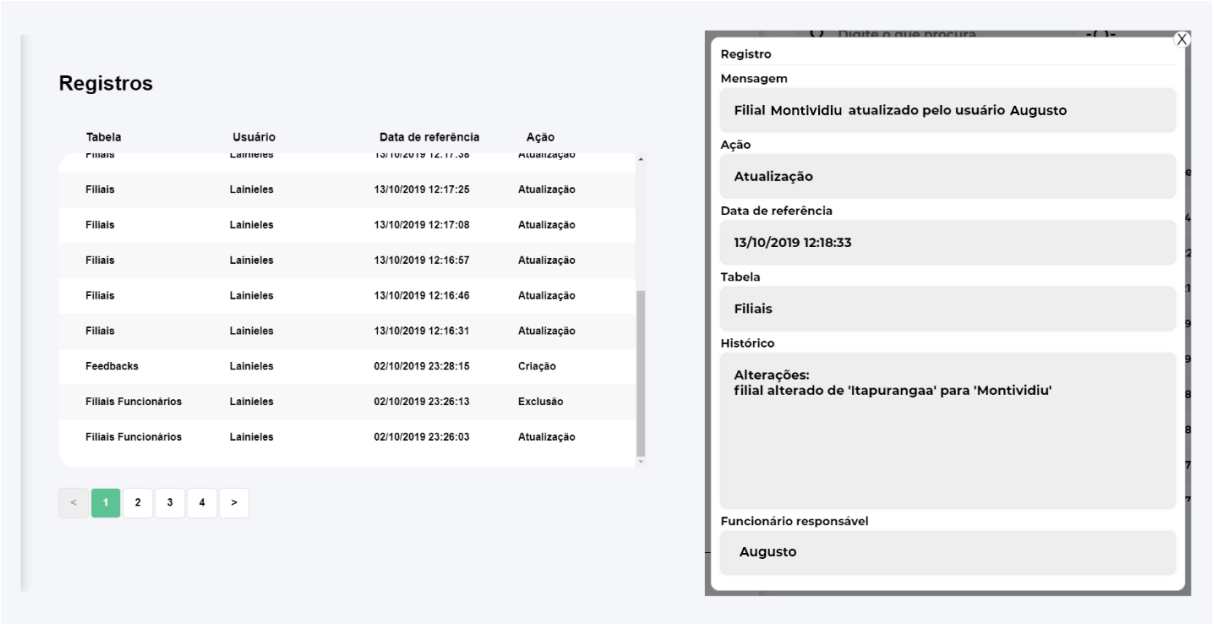
Tabela 18 – História (#18): Listagem de registro de atividades.

Identificação da história	**#18**
Título da história	Listagem de registro de atividades
Fonte da história	Análise do sistema
Data	14/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	

Como dono do produto desejo que seja implementada a página de listagem dos registros de atividades do sistema, que ao clicar em uma linha da tabela, será exibida uma janela flutuante exibindo todas as informações já exibidas na tabela, que serão a tabela de origem do registo, o usuário que realizou a ação, a data de referência, de quanto foi realizada aquela ação e a ação da atividade, ou seja, criação atualização ou exclusão. Além dessas informações, na janela flutuante será exibido a mensagem personalizada para o registro e o histórico das informações em casos de atualização.

As informações exibidas na página de listagem dos registros de atividades, foram detalhados na história que foi requisitada sua implementação, dentre elas está a tabela de origem, funcionário, data de referência e ação do registro, além da ação adicional, que ao clicar em uma linha da tabela, será exibida uma janela flutuante com informações mais detalhadas, como o histórico, em caso de ação de atualização e a mensagem personalizada do registro em questão, como demonstrado na Figura 14.

Figura 14 – Ilustração da página de listagem de registros de atividades do sistema.



Fonte: Própria.

3.3.3 MANTER INFORMAÇÕES

As páginas de manutenção das informações, tanto cadastro como edições seguirão as regras descritas em histórias, prototipação de interfaces já descritos anteriormente e regras definidas para o banco de dados, tanto pelo seu tipo quanto à restrição dos dados relacionados à obrigatoriedade dos mesmos.

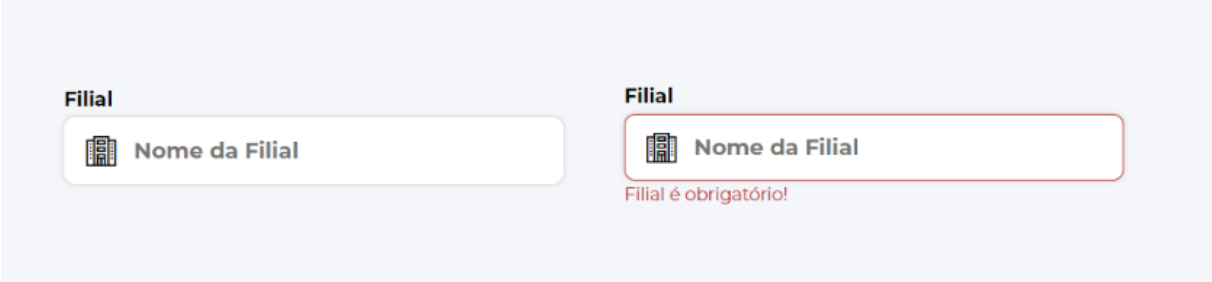
Foram definidos componentes principais para a construção desses formulários e telas, incluindo componentes de atualização para o usuário, informando se sua requisição obteve sucesso, ou não. Seu desenvolvimento se deu à partir da história a seguir.

Tabela 19 – História (#19): Componentes de formulário.

Identificação da história	**#19**
Título da história	Componentes de formulário
Fonte da história	Análise do sistema
Data	16/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	
Como desenvolvedor desejo que sejam implementados componentes padronizados para a construção das telas de manutenção das informações, tanto cadastros como edições, além de pequenas caixas de auto ajuda informando ao usuário se seu cadastro ou alteração foi realizada com sucesso ou não. Válido lembrar da necessidade da validação do formulário para as regras de todos os campos.	

Os padrões definidos para tais componentes foram pensados na facilidade e praticidade do usuário em identificar o campo a ser preenchido, de forma que será exibido uma legenda e um ícone representativo para o campo, além de um rótulo, caso aquele campo não tenha sido preenchido. Em caso de falha na validação deverá ser exibida uma mensagem informando o erro a qual ele foi submetido e sendo destacado dos demais com uma borda vermelha, indicando que aquele campo não foi preenchido corretamente, demostrados na Figura 15.

Figura 15 – Protótipos padronizados para os campos dos formulários.



Fonte: Própria.

Já o componente responsável por manter o usuário informado de suas ações apresenta-se um componente mais simples, apenas com uma mensagem com instruções e cores intuitivas de acordo com o resultado de sua requisição ao sistema, assim como na Figura 16.

Figura 16 – Componente responsável por manter o usuário informado das atualizações do sistema.

Fonte: Própria.

Definidos o modelo e características dos principais componentes que compõem as telas responsáveis por manter os dados do sistema, foram definidas histórias com suas respectivas regras de negócio, sejam elas para validações do formulário ou para inclusão de opções extras na interface.

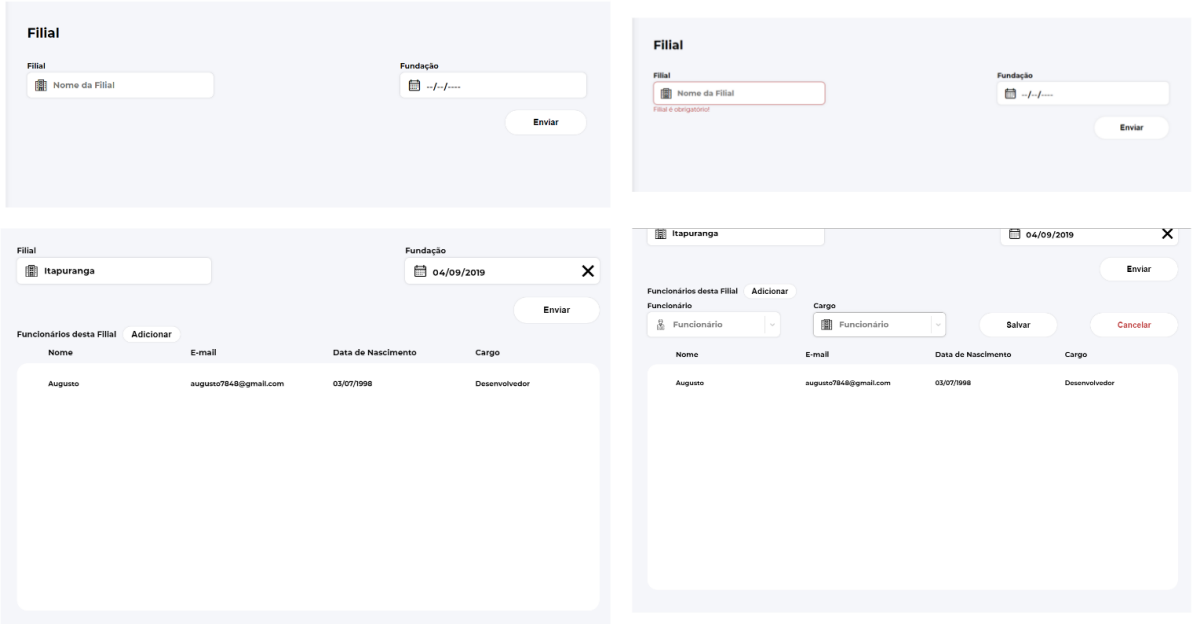
Tabela 20 – História (#20): Cadastro e edição de filiais.

Identificação da história	**#20**

Título da história	Cadastro e edição de filiais
Fonte da história	Análise do sistema
Data	18/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	
Como dono do produto desejo que seja implementada a tela de cadastro e edição de filiais, com os campos referentes ao nome e sua data de fundação, sendo que apenas o nome é obrigatório durante o preenchimento do formulário. Uma vez cadastrada, deve-se exibir uma tabela informando os funcionários alocados naquela filial e seus respectivos cargos.	

A história que define a implementação da tela de cadastro e edição de filiais foi definida e como regra principal a definição da obrigatoriedade do campo de nome, e na mesma tela, uma tabela responsável por listar todos os funcionários registrados na mesma com seus respectivos cargos, como demonstrado na Figura 17.

Figura 17 – Página de cadastro e edição de filiais.



Fonte: Própria.

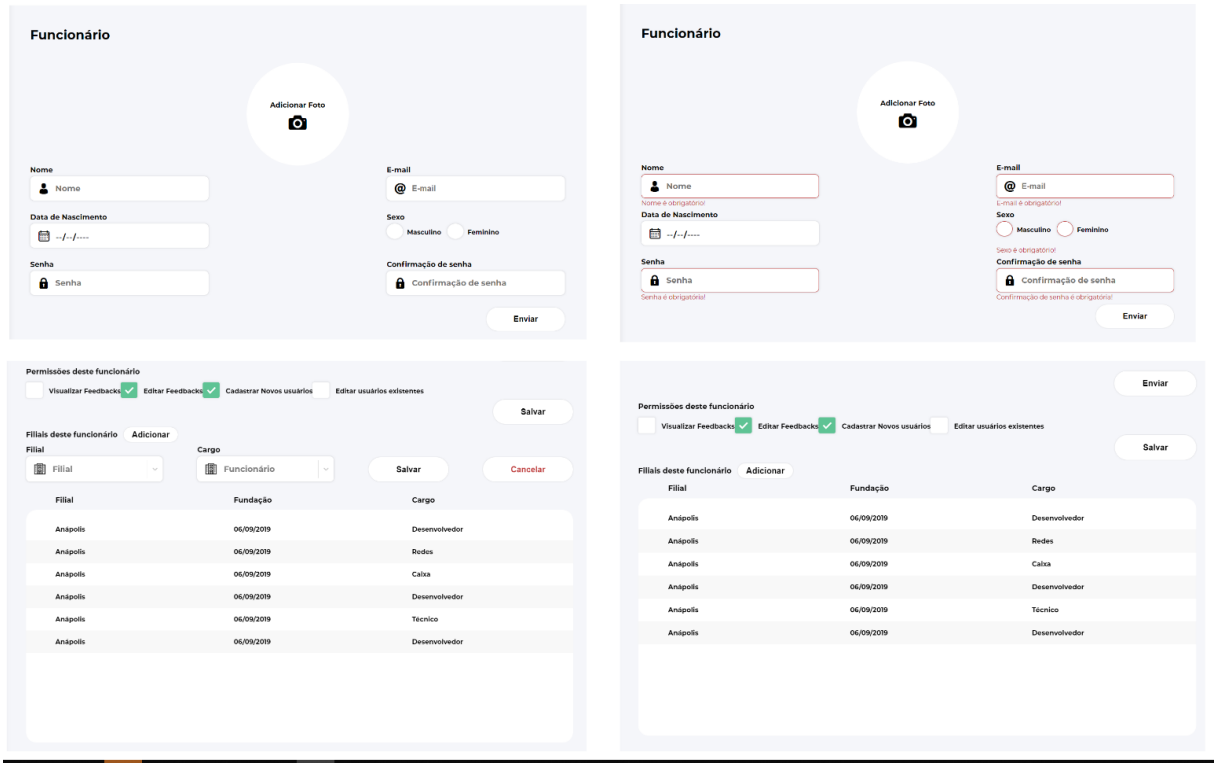
A ação de clicar no botão adicionar aciona o evento do formulário responsável por manter as relações entre as filiais e os funcionários, possibilitando a alocação de um funcionário à filial selecionada.

Tabela 21 – História (#21): Cadastro e edição de funcionários.

Identificação da história	**#21**
Título da história	Cadastro e edição de funcionários
Fonte da história	Análise do sistema
Data	20/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	
Como dono do produto desejo que seja implementada a tela de cadastro e edição de funcionários, como regras para a interface foi definido que os campos de nome, *e-mail*, sexo, e senha devem ser obrigatórios, levando em consideração que as senhas, para um novo cadastro devem ser as mesmas. Assim como a tela de cadastro e edição de filiais, deve-se exibir uma tabela indicando as filiais que aquele funcionário está alocado, com o acréscimo de um formulário à parte para gerenciamento das permissões daquele funcionário em questão.	

A construção da tela de cadastro de funcionários foi definida pela história anterior, foram definidos os campos obrigatórios, sendo eles o campo de nome, *email*, sexo e senhas iguais, demonstrados na Figura 18, no caso de novos cadastros.

Figura 18 – Página de cadastro e edição de funcionários.



Fonte: Própria.

Assim como o cadastro de filiais, possuirá uma tabela, porém informando as filiais que aquele funcionário está alocado, com o acréscimo do formulário que gerenciará as permissões do funcionário em questão dentro do sistema, caso esteja no modo de edição.

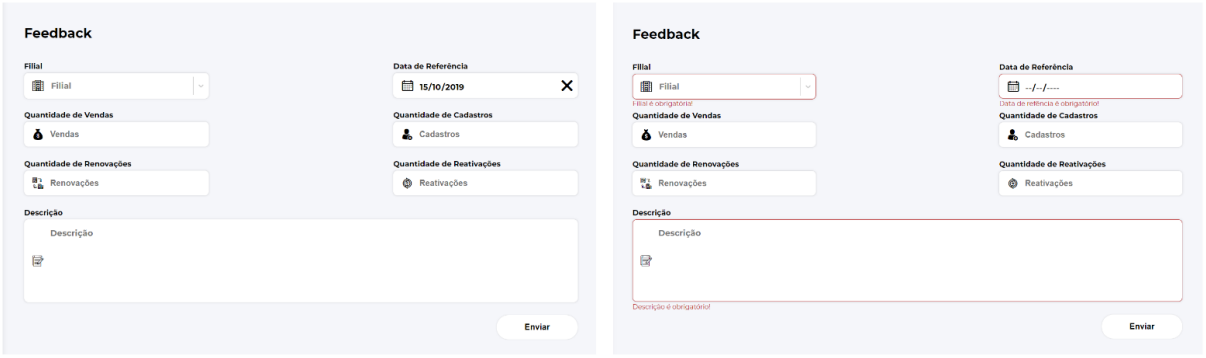
Tabela 22 – História (#22): Cadastro de *feedbacks*.

Identificação da história	**#22**
Título da história	Cadastro de *feedbacks*
Fonte da história	Análise do sistema
Data	23/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	

Como dono do produto desejo que seja implementada a tela de cadastro de *feedbacks*, e como regra para os campos do formulário, visto que, apenas a descrição, a filial e a data de referência do *feedback* serão obrigatórios, já que é possível um dia de serviço não realizar nenhum serviço proposto para o cadastro pelo sistema. É valido ressaltar que os *feedbacks* não poderão ser editados.

Dada a história, foi realizado a implementação do cadastro de *feedbacks*, considerando que apenas os campos de descrição, filial e data de referência ao *feedback* seriam obrigatórios, assim como demonstrado na Figura 19, levando-se em consideração que durante um dia inteiro de trabalho poderiam não serem realizadas as atividades propostas para o cadastro no sistema.

Figura 19 – Página de cadastro e edição de *feedbacks*.



Fonte: Própria.

Assim como definida na história, nenhuma informação adicional ou campo foram adicionados como nos demais módulos anteriores, visto que todas as informações a serem exibidas estão no próprio formulário.

3.3.4 DADOS E RELATÓRIOS

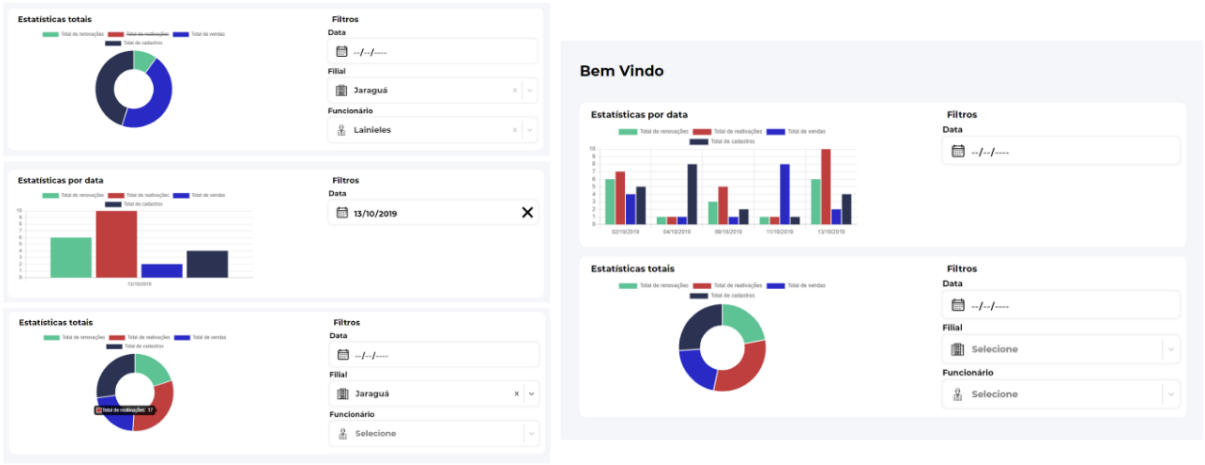
Definidas as regras de negócio de todas as principais telas e funcionalidades do sistema, é válido ressaltar o fundamento ao qual a construção desse projeto foi requisitada: a otimização dos processos e atividades diárias de uma empresa, visando controle, e melhoria de processos já realizados porém sem fim específico.

Tabela 23 – História (#23): Construção de gráficos e relatórios a partir dos dados cadastrados.

Identificação da história	**#23**
Título da história	Construção de gráficos e relatórios a partir dos dados cadastrados
Fonte da história	Análise do sistema
Data	28/07/2019
Responsável pela história	Daniel Moreira
Descrição da história	
Como dono do produto desejo que seja implementado e exibido na tela inicial do sistema, gráficos a respeito dos *feedbacks* realizados. O primeiro gráfico deverá apresentar um total de vendas, cadastros, renovações e reativações realizadas durante os dias de trabalho agrupados por dias, sendo exibidos até no máximo seis dias, com possibilidade de serem filtrados independentemente da data. O segundo gráfico deverá apresentar uma visão geral, com o resultado de todas as vendas, cadastros, renovações e reativações, sendo possível filtrar por data, filial e funcionário, os filtros poderão ser preenchidos independentemente e deverão funcionar em conjunto, para melhor visualização dos dados. Vale lembrar que ambos os gráficos deverão ter a possibilidade de ocultar ou exibir um dado específico, como ocultar apenas as vendas, ou apenas as renovações,e assim por diante.	

Como a proposta principal do projeto, é proporcionar dados palpáveis e relatórios que mostrem o desempenho das atividades diárias da empresa que o utilizará, a última história, mas não menos importante, é justamente sobre a criação de componentes ou páginas que permitem a visualização desses dados, demonstrados na Figura 20.

Figura 20 – Página principal do sistema com exibição dos gráficos e seus respectivos filtros.



Fonte: Própria.

Foi utilizada a biblioteca *react-chartjs2*, uma extensão da biblioteca *Chart.js*, amplamente utilizada para a construção de gráficos utilizando o *framework JQuery*, mas adaptada para o *ReactJS*.

4 TRABALHOS RELACIONADOS

Para se propor uma nova solução para um problema existente, deve-se analisar o mercado, saber como são as ferramentas, produtos ou técnicas abordadas por algumas empresas que trabalham ou se aproximam do problema específico que o projeto propõe solucionar. Como resultado das buscas por aplicações que se assemelham com a proposta desse projeto foram encontradas aplicações que serão analisadas ao longo deste capítulo.

4.1 *PULSES*

A ferramenta *Pulses*, acessível no sítio *https://www.pulses.com.br*, é um ensaio de uma rede social, onde há a promoção de uma interação entre colaboradores de uma empresa, visando transparência dos processos e colaboração.

O *Pulses*, trabalha mais especificamente como se fosse um fórum de dúvidas e sugestões, em que os colaboradores da equipe, como são chamados na plataforma, são incentivados a colaborar com os processos de outros membros, promovendo *rankings* (classificação dos membros) de colaboração, com notas de 0 a 10, julgando o quão prestativo, eficiente ou comunicativo é aquele colaborador, classificando-se também por cada um desses quesitos citados.

Assim como a proposta desse trabalho, é um sistema *web*, evitando a necessidade de instalação em qualquer máquina de qualquer usuário, fazendo-se acessível em qualquer plataforma, seja um computador, celular ou *tablet* através de um navegador e acesso à internet.

Como desvantagem e o que difere de fato desse trabalho, é o foco nos colaboradores. A ferramenta *Pulses*, possui um enfoque no colaborador, disponibilizando *rankings* e demonstrando o quão eficiente ou produtivo é um colaborador individualmente dentre a equipe e não a gestão dos processos em si.

4.2 *SOFTWARE* AVALIAÇÃO

O *Software* Avaliação, disponível em *https://www.softwareavaliacao.com.br/* é mais uma ferramenta com o enfoque empresarial e avaliação de colaboradores. Visando desempenho, a plataforma disponibiliza uma série de avaliações para colaboradores responsáveis por avaliar o desempenho e produtividade de um colaborador, fazendo assim, com que seja possível

determinar prazos e metas para atividades dos colaboradores da empresa que utiliza esse serviço.

Diferente da aplicação anterior, onde há a interação de colaboradores, as avaliações são realizadas pela gerência ou gestão da empresa, fazendo com que o perfil do colaborador seja traçado por olhos superiores e consequentemente, a lapidação das habilidades de um colaborador, extraindo valor para a empresa e gerando *feebacks* bem elaborados pelo próprio sistema para os gestores, sobre os colaboradores promovendo reconhecimento para os melhores colaboradores.

Mais uma vez, a ferramenta demonstrou ser focada nos colaboradores, visando um olhar dos gestores e não dos colaboradores sobre eles mesmos e sobre a empresa, tornando visível outra desvantagem que seria a manuseabilidade do *software* pelos gestores e não dos colaboradores, o que pode causar em favoritismo e falha na avaliação do desempenho real dos colaboradores.

4.3 *TRELLO*

O *Trello*, disponível em *https://trello.com/* também pode ser considerado um *software* de gestão de processos, com um foco um tanto quanto diferente dos demais apresentados. Seu enfoque principal é realmente a gestão de processos com a possibilidade de criação de listas de atividades ou organização em colunas, listas e categorias, no *software*, identificadas por *tags* (Etiquetas), mais especificamente como o Quadro de *Kanban* funciona, como já exemplificado neste trabalho.

Diferente dos outros *softwares* aqui citados, o *Trello* possui planos gratuitos para times, independente da quantidade de colaboradores, limitando-se apenas à quantidade de quadros, ou projetos por assim dizer, disponíveis em sua versão gratuita, não fazendo excepção das funcionalidades principais do sistema.

Com a interface enxuta e bastante intuitiva o *Trello* se faz uma plataforma de fácil manuseio e interatividade para qualquer usuário e focando na produtividade, sem longos formulários ou rodeios. O que se enquadra também como desvantagem, visto que as informações são apenas textos, e não uma informação padronizada de fato, não há como extrair informação de um processo executado por determinado colaborador.

4.4 *PIPEFY*

Pipefy, disponível em https://www.pipefy.com/ também pode ser considerado um software para gerenciamento empresarial e de *feedbacks* dos colaboradores, com uma interface bastante intuitiva, cores chamativas diversas outras ferramentas, mais especificamente, focadas para empresas de desenvolvimento de software ou que trabalham com o Quadro de *Kanban* junto ao *Scrum*.

Essa ferramenta, em seu módulo de gerenciamento de atividades se assemelha muito ao *Trello*, por conter quadros organizáveis e reordenáveis, torna a organização das tarefas muito mais fácil e intuitiva.

Já em seu módulo de *feedbacks*, que é justamente a proposta desse trabalho, mais se assemelha com o módulo anterior porém de forma estática, com colunas de nomes fixos, como "em andamento, concluído", entre outros, demonstrando a aplicabilidade em processos, porém não extraindo de forma coesa e precisa as informações desenvolvidas nos processos e sim os processos como um todo em seu estado atual.

5 CONSIDERAÇÕES FINAIS

Foi descrito todo o processo de desenvolvimento do projeto utilizando o *Scrum*, uma metodologia ágil para gestão, planejamento e desenvolvimento de *software*, visando agilidade e qualidade na entrega do produto final.

Assim como é incentivado pelo *Scrum*, o planejamento e estrutura das principais funcionalidades do *software* foram evoluindo e se alterando dada a necessidade de alterações, comprovando a dinamicidade de se desenvolver usando esse *framework*, que visa mais em menos tempo sem perca na qualidade.

O desenvolvimento do *software* foi concluído com excelência, atendendo a todas as histórias definidas, sendo alternadas apenas pela perspectiva de visão e não apenas por pessoa em si. Antes e durante a etapa de desenvolvimento do *software*, estando apto a ser utilizado. Assim como já citado anteriormente, foi utilizado um sistema de controle de versão, mais especificamente o *Github*, sendo assim, todas as etapas do desenvolvimento foram documentadas no próprio repositório, e são de fácil acesso caso necessário ou para possíveis consultas de todas as versões do *software*. Vale lembrar que o projeto foi desenvolvido durante 30 dias, divididos em três *sprints* de dez dias cada e um fluxo de trabalho de 4 horas diárias, desde a fase de planejamento à fase de desenvolvimento.

Uma das principais dificuldades durante o processo de desenvolvimento do software quanto à parte do servidor, foi a adaptação aos *frameworks* utilizados com a integração ao banco de dados, por se tratar de um *ORM*, necessitou de aprendizado das regras impostas pelo *framework*.

Outra dificuldade durante o processo de desenvolvimento foi relacionado ao desenvolvimento das interfaces, que como foi proposto, deveriam ser interfaces amigáveis, intuitivas e modernas, para fácil usabilidade dos usuários, o que requiriu criatividade uso de interfaces de *softwares* relacionados.

Ademais, uma etapa importantíssima para a conclusão do objetivo foram os gráficos implementados na tela principal do sistema, visto que eram fundamentais para o controle dos processos desempenhados pelos funcionários da empresa que o utilizará.

Para trabalhos futuros, é sugerido o desenvolvimento desta mesma aplicação com outra linguagem de programação e outra metodologia de desenvolvimento, a fim de comparar os processos de desenvolvimento e também a dinamicidade e tempo de planejamento e execução do projeto.

REFERÊNCIAS

Dirk Krafzig, Karl Banke, D. S. **Enterprise SOA: Service-Oriented Architecture Best Practices**. New Jersey: Prentice Hall, 2004. 408 p.

FACEBOOK. **ReactJS**. 2019. Disponível em: <https://reactjs.org/>. Acesso em: 20 de jun. 2019.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. 162 p. Tese (Doutorado) — UNIVERSITY OF CALIFORNIA, 2000.

GITHUB. **Github**. 2019. Disponível em: <https://github.com/>. Acesso em: 12 de jun. 2019.

GOOGLE. **V8 Engine**. 2019. Disponível em: <https://v8.dev/>. Acesso em: 26 de jun. 2019.

MOURA, R. A. **Kanban: a simplicidade do controle da produção**. 6ª edição. ed. Brasil: Imam, 2003. 230 p.

MOZILLA. **Mozilla**. 2019. Disponível em: <https://developer.mozilla.org/pt-BR/>. Acesso em: 14 de jun. 2019.

NODE.JS. **Node.js**. 2019. Disponível em: <https://nodejs.org/pt-br/>. Acesso em: 23 de jun. 2019.

OSMANI, A. **Learning JavaScript Design Patterns**. 1.7.0. ed. [s.n.], 2017. Disponível em: <https://addyosmani.com/resources/essentialjsdesignpatterns/book/>. Acesso em: 18 de jun. 2019.

OVERFLOW, S. **Stack Overflow**. 2019. Disponível em: <https://pt.stackoverflow.com/>. Acesso em: 21 de jun. 2019.

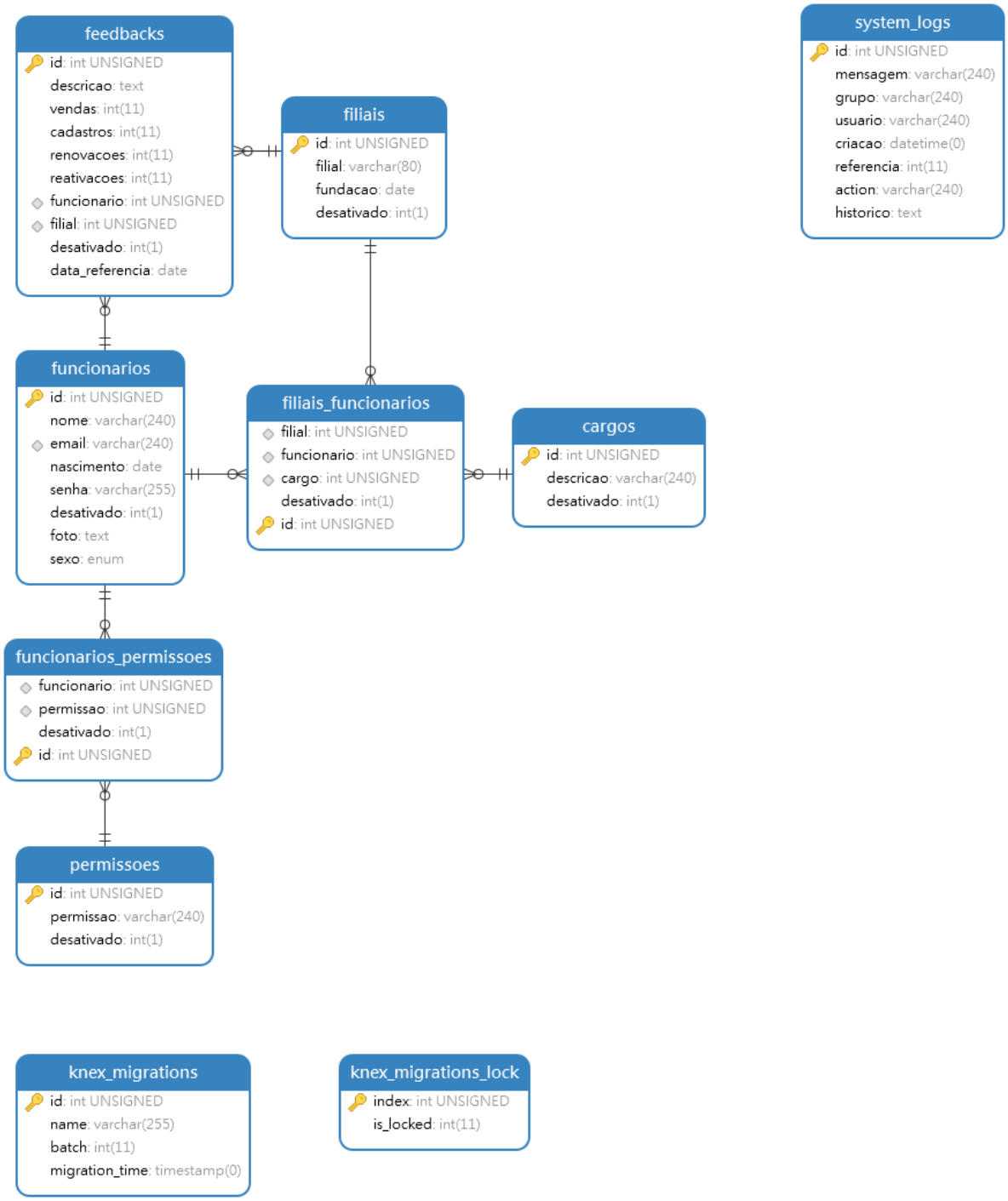
SOMMERVILLE, I. **Engenharia de Software**. 9ª edição. ed. São Paulo: PEARSON Prentice Hall, 2003. 530 p. ISBN 9788579361081.

SUTHERLAND, J. **SCRUM - A arte de fazer o dobro do trabalho na metade do tempo**. 1ª. ed. São Paulo: LeYa, 2014. 158 p.

TELEGRAM. **Telegram**. 2019. Disponível em: <https://telegram.org/>. Acesso em: 16 de jun. 2019.

APÊNDICE A – MODELO RELACIONAL - DER VERSÃO FINAL.

Figura 21 – Modelo relacional do banco de dados - DER Versão final.



Fonte: Própria.

APÊNDICE B – DICIONÁRIO DE DADOS

Tabela 24 – Tabela de permissões

Tabela: permissoes					
Campo Físico	Chaves	Tipo	Tamanho	Nulo	Descrição
id	PK	INT		Not Null	Código da permissão.
permissao		VARCHAR	240	Null	Descrição da permissão.
desativado		INT	1	Null	Coluna que identifica se um registro está oculto ou não.

Tabela 25 – Tabela de cargos

Tabela: cargos					
Campo Físico	Chaves	Tipo	Tamanho	Nulo	Descrição
id	PK	INT		Not Null	Código do cargo.
descricao		VARCHAR	240	Null	Descrição do cargo.
desativado		INT	1	Null	Coluna que identifica se um registro está oculto ou não.

Tabela 26 – Tabela de filiais

Tabela: filiais					
Campo Físico	Chaves	Tipo	Tamanho	Nulo	Descrição
id	PK	INT		Not Null	Código da filial.
descricao		VARCHAR	240	Null	Descrição da filial.
fundacao		DATE		Null	Data de fundação da filial.
desativado		INT	1	Null	Coluna que identifica se um registro está oculto ou não.

Tabela 27 – Tabela de funcionários

Tabela: `funcionarios`					
Campo Físico	Chaves	Tipo	Tamanho	Nulo	Descrição
id	PK	INT		Not Null	Código do funcionário.
nome		VARCHAR	240	Null	Nome do funcionário.
email		VARCHAR	240	Not Null	E-mail do funcionário.
nascimento		DATE		Null	Data de nascimento do funcionário.
senha		VARCHAR	255	Null	Senha do funcionário.
foto		TEXT		Null	Nome do arquivo único da foto no servidor.
sexo		ENUM	1	Null	Sexo do funcionário.
desativado		INT	1	Null	Coluna que identifica se um registro está oculto ou não.

Tabela 28 – Tabela de *feedbacks*

Tabela: `feedbacks`					
Campo Físico	Chaves	Tipo	Tamanho	Nulo	Descrição
id	PK	INT		Not Null	Código do *feedback*.
descricao		TEXT		Null	Descrição do *feedback*.
vendas		INT	11	Null	Quantidade de vendas.
cadastros		INT	11	Null	Quantidade de cadastros.
renovacoes		INT	11	Null	Quantidade de renovações.
reativacoes		INT	11	Null	Quantidade de reativações.
data_referencia		DATE		Null	Data de referencia do *feedback*.
funcionario	FK	INT		Not Null	Identificador do funcionário responsável.
filial	FK	INT		Not Null	Identificador da filial do *feedback* em questão.
desativado		INT	1	Null	Coluna que identifica se um registro está oculto ou não.

Tabela 29 – Tabela de relação entre filiais e funcionários

Tabela: `filiais_funcionarios`					
Campo Físico	Chaves	Tipo	Tamanho	Nulo	Descrição
id	PK	INT		Not Null	Código da relação.
funcionario	FK	INT		Not Null	Identificador do funcionário.
filial	FK	INT		Not Null	Identificador da filial.
cargo	FK	INT		Not Null	Identificador do cargo.
desativado		INT	1	Null	Coluna que identifica se um registro está oculto ou não.

Tabela 30 – Tabela de relação entre funcionários e permissões

Tabela: `funcionarios_permissoes`					
Campo Físico	Chaves	Tipo	Tamanho	Nulo	Descrição
id	PK	INT		Not Null	Código da relação.
funcionario	FK	INT		Not Null	Identificador do funcionário.
permissao	FK	INT		Not Null	Identificador da permissão.
desativado		INT	1	Null	Coluna que identifica se um registro está oculto ou não.

Tabela 31 – Tabela de registros do sistema

Tabela: `system_logs`					
Campo Físico	Chaves	Tipo	Tamanho	Nulo	Descrição
id	PK	INT		Not Null	Código do registro.
mensagem		VARCHAR	240	Null	Mensagem do registro.
grupo		VARCHAR	240	Null	Tabela de referência do registro.
usuario		VARCHAR	240	Null	Identificador do usuário.
referencia		INT	11	Null	Identificador do registro em sua respectiva tabela.
historico		TEXT		Null	Histórico dos dados.
criacao		DATE		Null	Data de criação do registro.
action		VARCHAR	240	Null	Identifica a ação realizada.