UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO INE
CURSO SISTEMAS DE INFORMAÇÃO

Bernardo Schveitzer de Souza e Otávio Carneiro Ribeiro

S.C.O.U.T - SISTEMA DE CONTROLE ORÇAMENTÁRIO E UTILIZAÇÃO DE TEMPO

Florianópolis
2021

Bernardo Schveitzer de Souza e Otávio Carneiro Ribeiro

S.C.O.U.T - SISTEMA DE CONTROLE ORÇAMENTÁRIO E UTILIZAÇÃO DE TEMPO

Trabalho Conclusão do Curso de Graduação em Sistemas de Informação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Bacharel em Sistemas de Informação.
Orientador: Prof. Martín Augusto Gagliotti Vigil, Dr.

Florianópolis

2021

Ficha de identificação da obra

Ribeiro, de Souza, Otávio Carneiro, Bernardo Schveitzer
S.C.O.U.T - SISTEMA DE CONTROLE ORÇAMENTÁRIO E UTILIZAÇÃO DE TEMPO / Otávio Carneiro Ribeiro, Bernardo Schveitzer de Souza; orientador, Martín Augusto Gagliotti Vigil, 2021.
60 p.
Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Sistema de Informação, Florianópolis, 2021.
Inclui referências.
1. Sistema de Informação. 2. Ponto eletrônico. 3. Gerencia de projetos. 4. Aplicação Web. 5. Jornada de trabalho. I. de Souza, Bernardo Schveitzer. II. Vigil, Martín Augusto Gagliotti. III. Universidade Federal de Santa Catarina. Graduação em Sistema de Informação. IV. Título.

Bernardo Schveitzer de Souza e Otávio Carneiro Ribeiro

S.C.O.U.T - SISTEMA DE CONTROLE ORÇAMENTÁRIO E UTILIZAÇÃO DE TEMPO

Este Trabalho Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Sistemas de Informação" e aprovado em sua forma final pelo Curso de Sistemas de Informação

Florianópolis, 27 de agosto de 2021.

_____

Prof. Cristian Koliver, Dr.
Coordenador do Curso

_____

Prof. Martín Augusto Gagliotti Vigil, Dr.
Orientador

Banca Examinadora:

_____

Prof. Fábio Rodrigues de La Rocha, Dr.
Avaliador

_____

Renan Luiz Arceno
Avaliador

Este trabalho é dedicado aos nossos queridos familiares, amigos e ao Professor Leandro José Komosinski.

AGRADECIMENTOS

Gostaríamos de agradecer a todos os professores e servidores da UFSC, Universidade Federal de Santa Catarina que diretamente ou indiretamente nos ajudaram na nossa jornada acadêmica. Estendemos os nossos agradecimentos ao LabTic, Laboratório de Tecnologia de Informação e Comunicação da UDESC, Universidade do Estado de Santa Catarina que nos deu a oportunidade do primeiro estágio na área de desenvolvimento de software. E por fim, agradecer aos nossos colegas, amigos, em especial ao grupo FDR, e familiares que nos incentivaram a concluir este ciclo das nossas vidas. Nosso muitíssimo obrigado.

“O tempo dura bastante para aqueles que sabem aproveitá-lo.” (Leonardo Da Vinci)

RESUMO

O ponto eletrônico é um dispositivo utilizado por empresas para controlar a hora da entrada e saída da jornada de trabalho de seus colaboradores. Além de registrar estas horas, ele auxilia na administração das mesmas, aperfeiçoando as rotinas de trabalho. O gerenciador de projetos também é um instrumento que torna as horas trabalhadas ainda mais eficientes, porém, não da mesma forma que o ponto eletrônico. Com isso, se teve a ideia de unir as duas ferramentas para que se possa otimizar ainda mais uma jornada de trabalho. E com a falta de um produto que realiza isso no mercado, foi decidido desenvolver um software que cumpre este objetivo. Neste trabalho de conclusão de curso é proposto um sistema de controle de jornada de trabalho com um sistema gerenciador de projetos integrado em um único produto, denominado inicialmente como "SCOUT", que significa Sistema de Controle Orçamentário e Utilização de Tempo. O sistema foi projetado, desenvolvido e testado. Para atingir os objetivos propostos neste trabalho, foi utilizada a metodologia conhecida por *Design Science Research Methodology* (DSRM). O sistema foi desenvolvido utilizando linguagens de programação para aplicações *Web*. E os testes provam que o sistema é funcional, é viável e tem potencial para ser um produto.

Palavras-chave: Ponto eletrônico, jornada, controle, horas, trabalho, gerência, projetos.

ABSTRACT

The electronic worksheet is a device used by companies to control the entry and exit time of their employees' working hours. Beside recording these hours, it helps to manage them, improving the work routines. The projects manager is also an instrument that makes the hours worked even more efficient, but not in the same way as the electronic worksheet. This way the idea was to unite the two tools so that a workday can be further optimized. And with the lack of a product that accomplishes this in the market, it was decided to develop software that reached this goal. The present final paper is a workday control system with an integrated project manager tool in a unique product, first called "SCOUT". The software was designed, developed and tested. To achieve the objectives proposed in this project, the methodology known as Design Science Research Methodology (DSRM) was used. The system was developed using programming languages for Web applications. And the tests prove that the system is functional, viable and has the potential to be a product.

Keywords: Electronic worksheet, control, hours, work, management, projects.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SCOUT Sistema de Controle Orçamentário e Utilização de Tempo
UFSC Universidade Federal de Santa Catarina
UDESC Universidade do Estado de Santa Catarina
LabTic Laboratório de Tecnologia de Informação e Comunicação
DSRM *Design Science Research Methodology*
ROI *Return of Investment*
HTML *HyperText Markup Language*
CSS *Cascading Style Sheets*
SQL *Structured Query Language*
REST *Representational State Transfer*
API *Application Programming Interface*
SPA S*ingle-page Application*
HTTP *Hypertext Transfer Protocol*
UI *User Interface*
UX *User Experience*
UML *Unified Modeling Language*
WWW *World Wide Web*
JWT *JSON Web Token*
JSON *JavaScript Object Notation*

SUMÁRIO

1 INTRODUÇÃO

Com a necessidade das empresas começarem a armazenar os horários de entrada, saída e até mesmo as pausas para o almoço de seus colaboradores, foi criado o relógio ponto, dispositivo mecânico que armazena estes horários em pedaços de papel pertencentes a cada colaborador. Com o passar do tempo, Barros (2019) explica que estes dispositivos começaram a se tornar sistemas informatizados e com isso o seu nome popular mudou para ponto eletrônico. Hoje eles são usados com registros marcados por biometria e até mesmo reconhecimento facial. Além disso, é possível armazenar os seus dados de maneira centralizada, podendo tirar conclusões de como realizar uma jornada de trabalho mais eficiente e ter um melhor controle destas horas.

Outra ferramenta que busca ajudar empresas a tornar as suas horas trabalhadas ainda mais eficientes é o gerenciador de projetos. Também na maioria das vezes eletrônico, o gerenciador de projetos auxilia no planejamento de cada projeto a fim de cumprir seus requisitos, alocando recursos e a programando suas atividades. Alguns exemplos deles são: Jira (ATLASSIAN, 2021), Trello (ATLASSIAN, 2021), Microsoft Project (MICROSOFT, 2021) e Podio (CITRIX SYSTEMS, 2021).

Apesar da grande ajuda que os gerenciadores de projetos trouxeram às empresas e até mesmo para pessoas que só querem administrar melhor as suas tarefas, um dos seus grandes problemas é a falta da funcionalidade de poder monitorar a quantidade de horas gastas para realização de cada atividade. Esse problema pode acarretar em erros para estipular o prazo da conclusão do projeto e também pode gerar gastos altos e desnecessários. Com isso, se foi pensado em planejar e desenvolver um sistema integrado de controle de jornada de trabalho (ponto eletrônico) e gerência de projetos de um jeito sucinto, de fácil utilização e que não exija uma grande curva de aprendizado dos usuários. Denominado inicialmente como “SCOUT”, que significa Sistema de Controle Orçamentário e Utilização de Tempo. Neste sistema os colaboradores e seus gerentes teriam um acompanhamento em “tempo real” do projeto, mostrando os custos e esforços que já foram gastos em cada atividade, tarefa ou módulo. Podendo também gerar relatórios para acompanhar o projeto por inteiro, através do número total de horas gasta no projeto, gasta por colaborador e por determinada atividade. Com a união destes dois tipos de sistemas o usuário teria um maior controle e organização de suas horas trabalhadas, deixando-o mais seguro em relação aos problemas citados anteriormente.

2 OBJETIVOS

2.1 OBJETIVO GERAL

Projetar e desenvolver um sistema computacional com a funcionalidade de um gerenciador de projetos baseado na metodologia SCRUM integrado com um "ponto eletrônico", denominado pela equipe de SCOUT.

2.2 OBJETIVOS ESPECÍFICOS

Para atingir o objetivo geral, os objetivos específicos deste trabalho são:

1. Identificar os requisitos funcionais do sistema.
2. Desenvolver um sistema web implementando os requisitos coletados.
3. Realizar testes funcionais para validar se o sistema cumpre os seus requisitos.

3 METODOLOGIA

Para atingir os objetivos deste projeto será utilizada a metodologia conhecida por *Design Science Research Methodology* (DSRM) que é um método de projetar artefatos para resolver problemas. Tal metodologia contém as seguintes etapas. A primeira etapa trata da identificação do problema e sua causa, que já foi executada e documentada na introdução deste projeto. A segunda etapa é onde definem-se os objetivos da solução. Em seguida, é projetado e desenvolvido um artefato para solucionar o problema proposto. Depois, na quarta etapa, o artefato é demonstrado para solucionar o problema e por fim, na quinta etapa, será avaliado para garantir que atinja os objetivos da segunda etapa (JUNIOR et al., 2017). Nas próximas subseções, serão descritas cada uma das etapas citadas acima, excluindo a primeira, uma vez que a mesma foi descrita anteriormente.

3.1 DEFINIR OBJETIVOS DA SOLUÇÃO

O objetivo principal da solução é o desenvolvimento de um software que une as funcionalidades de uma ferramenta de ponto eletrônico com um gerenciador de projetos. A definição dessa solução foi iniciada buscando referências do estado da arte de aplicações de ponto eletrônico e de gerência de projetos, focando nas que possibilitam um uso fácil, com interface amigável e configuração intuitiva. Após isso será estudado e decidido quais tecnologias serão utilizadas para a produção do artefato. E para finalizar, será executada uma pesquisa de mercado para coletar opiniões relacionadas ao problema levantado e a solução proposta.

3.2 PROJETAR E DESENVOLVER O ARTEFATO

Será criado um artefato de software para solucionar o problema. Primeiro será feita a análise de requisitos do projeto, com ela, será identificada todas as funcionalidades que o artefato deverá ter. Depois será levantado os tipos de usuário que existirão no software e as suas permissões. Em seguida será feito o protótipo das telas que ajudarão a entender e desenvolver os fluxos do sistema. Após finalizar essas etapas, será iniciada a implementação do artefato baseado nos requisitos e protótipos concebidos anteriormente. A implementação será dividida

em duas fases, o desenvolvimento do servidor e desenvolvimento da interface web para usuários.

3.3 DEMONSTRAR O USO DO ARTEFATO PARA SOLUCIONAR O PROBLEMA

Serão realizados testes para validar se o artefato resolve o problema conforme o planejado e se cumpre os seus requisitos. Os testes serão do tipo funcionais e serão executados manualmente. Primeiramente, será testada cada parte do artefato individualmente e depois todas as partes juntas como um todo. Com isso, será capaz de medir a qualidade e a confiabilidade do artefato.

4 FUNDAMENTAÇÃO TEÓRICA

O sistema que será desenvolvido se baseia na metodologia ágil SCRUM pelo fato de muitas empresas da atualidade usarem este modelo para gestão de seus projetos (SHASTRI, 2021). O desenvolvimento web foi escolhido para a implementação pela sua acessibilidade, tendo vista que a grande maioria dos colaboradores de uma empresa começam a sua jornada de trabalho acessando uma página web. Já as linguagens, frameworks e bibliotecas utilizadas no desenvolvimento do sistema foram decididas por serem tecnologias atuais no mercado, por serem fáceis de dar manutenção, por terem um tempo de desenvolvimento rápido e código limpo.

4.1 SCRUM

Scrum é um modelo ágil para gestão de projetos de software. A criação desse modelo se deu na indústria automobilística (Takeuchi & Nonaka, 1986), já na área de desenvolvimento de software, o *Scrum* começou a ficar popular com o trabalho de Schwaber, apresentado de forma completa e sistemática no livro de Schwaber e Beedle (2001).

O modelo ágil possui três perfis importantes para o seu funcionamento:

- *Scrum master*: geralmente é a pessoa que mais tem conhecimento sobre *Scrum* do time, é o facilitador. As reuniões são geridas por ele, é responsável por potencializar o trabalho da equipe e garantir que todos entendam o que é necessário ser feito e também é o principal resolvedor de conflitos.

- *Product owner*: a pessoa responsável pelo projeto em si. O *product owner* tem, entre outras atribuições, a de indicar quais são os requisitos mais importantes a serem tratados em cada sprint. O *product owner* é o responsável pelo ROI (*Return Of Investment*), e também por conhecer e avaliar as necessidades do cliente.

- *Scrum team*: é a equipe de desenvolvimento. Essa equipe não é necessariamente dividida em papéis como analista, designer e programador, mas todos interagem para desenvolver o produto em conjunto. Usualmente são recomendadas equipes de 6 a 10 pessoas.

Na estrutura do *Scrum* está presente o *product backlog*. É nele onde se encontram as funcionalidades a serem implementadas no projeto em forma de lista, não precisando necessariamente estar completo no início do projeto. Pode-se iniciar o *product backlog* apenas

com as funcionalidades mais evidentes e à medida que o projeto avança, se é tratado as novas funcionalidades que vão sendo descobertas. Apesar de começar com as funcionalidades mais evidentes, não significa que se deve fazer um levantamento inicial superficial, deve-se juntar o maior número de informações que se necessita.

Para melhor se administrar o tempo, é usado o *sprint.* O termo é usado para identificar um o ciclo de desenvolvimento, que geralmente possui de duas semanas a um mês de duração.

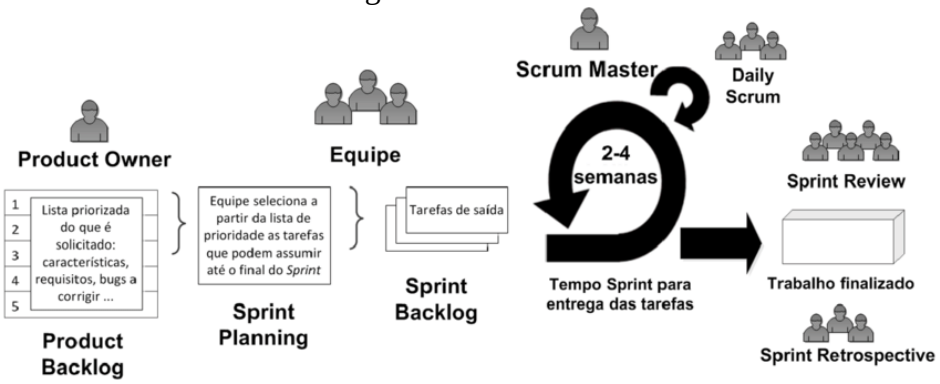
No início de cada *sprint* é feito um *sprint planning meeting*, no qual a equipe prioriza os elementos do *product backlog* a serem implementados. Tais elementos são denominados *tasks* que são atividades ou tarefas a serem feitas no projeto. A seguir, as *tasks* são transferidas do *product backlog* para o *sprint backlog*, ou seja, a lista de tarefas a serem implementadas no ciclo que se inicia.

A equipe se compromete a desenvolver as *tasks*, e o *product owner* se compromete a não trazer novas *tasks* durante o mesmo *sprint*. Se novas funcionalidades forem descobertas, serão abordadas em *sprints* posteriores.

O *sprint backlog* apresenta uma visão desses requisitos de forma mais voltada à maneira como a equipe vai desenvolvê-los.

Para desenvolver o SCOUT será utilizado o *Scrum* como metodologia, as *sprints* terão duração de duas semanas, porém podendo variar de acordo com o *sprint backlog* (WAZLAWICK, 2013).

Figura 1 – Ciclo do Scrum



Fonte: (Gouvea et al., 2016)

4.2 DESENVOLVIMENTO WEB

Desenvolvimento web é tudo aquilo que é projetado, implementado e utilizado dentro do ambiente da grande rede de computadores, a famosa *World Wide Web* (WWW). Sendo assim, o projeto fará parte da rede, contando com uma página web na parte do cliente e uma aplicação web na parte do servidor. A parte do cliente, é conhecida por *front-end,* onde está localizada a interface gráfica e é onde o usuário irá interagir com a aplicação. Já a parte do servidor é conhecida por *back-end* e não é vista pelo usuário. Nela é desenvolvida a lógica das regras de negócio e também é modelado o banco de dados. Para explicar melhor como será o desenvolvimento web desse projeto, é necessário separar em alguns pontos chave a seguir.

4.3 LINGUAGENS PARA DESENVOLVIMENTO WEB

Para suprir as necessidades da aplicação e melhor desempenho no ambiente web, a linguagem foi escolhida usando padrões conhecidos pela equipe de desenvolvimento, priorizando técnicas recentes com bons índices de utilização e desempenho. Por se tratar de uma aplicação web, a linguagem escolhida foi o JavaScript (HAVERBEKE, 2018), já que possui integração completa com o navegador, uma comunidade muito ativa, bibliotecas e ferramentas úteis para a aplicação. Além disso, mantém um padrão com a linguagem utilizada no servidor, que também utilizará JavaScript por meio do Node.js (OPENJS, 2021).

O JavaScript é uma linguagem de programação interpretada e estruturada, de script de alto nível, com tipagem fraca e multiparadigma. É usada globalmente em páginas da web e recentemente tem ganhando força em servidores web por meio do Node.js. Por ser multiparadigma, esse projeto utilizará tanto uma parte funcional como uma parte imperativa da linguagem. A linguagem surgiu no final do ano de 1995, porém só tomou força como referência entre as linguagens com a chegada do ECMAScript 5 no final de 2009. Os ECMAScripts são versões de especificações padrões criados juntamente com a ECMA *International*.

Para tornar a escrita de código em JavaScript mais eficiente, foi criado um conjunto de ferramentas denominado TypeScript (MICROSOFT, 2021). Nele foram adicionados recursos que não estão presentes na JavaScript nativo e que facilitam a criação de aplicações complexas. Exemplos destes recursos são a tipagem estática e a possibilidade de descoberta e

correção de erros durante o desenvolvimento, por ele conseguir apontar pontos de melhoria e problemas de compilação.

Para o hipertexto, estrutura das páginas Web composta por elementos e textos conectados entre si que criam o conjunto de informações mostrados ao usuário, será usado a linguagem Html (DUCKETT, 2014). Que corresponde a *Hiper Text Markup Language* (Linguagem de Marcação de Hipertexto) e serve para estruturar todo o conteúdo da parte da interface.

Com a estrutura da interface pronta, será usado CSS (DUCKETT, 2014) para estilizar a mesma. CSS corresponde a *Cascading Style Sheets* (Folhas de Estilo em Cascata) e é uma linguagem de estilo que é utilizada em conjunto a uma linguagem de marcação para deixá-la mais apresentável. O CSS é responsável pela aparência da página, permitindo customizar cores, posicionamento e formato de elementos, tudo para que ela se torne visualmente mais agradável possível.

4.4 NODE.JS

Para que se consiga trabalhar com JavaScript tanto no *front-end* quanto no *back-end* (*full-stack*) será utilizado o NodeJs. O mesmo é caracterizado por ser um ambiente de execução JavaScript que permite o programador criar a sua aplicação sem depender de um servidor, sistema operacional ou *browser* específicos. Além disso, oferece alta escalabilidade e flexibilidade (OPENJS, 2021), reduzindo os custos e aumentando as vantagens para a equipe de desenvolvimento.

4.5 MONGODB

Para o banco de dados será utilizado o MongoDB , ferramenta *opensource* flexível e com uma boa performance. O MongoDB é do tipo noSQL, ou seja, ele é capaz de armazenar dados não relacionais. Por ser um noSQL, ele possui um desempenho melhor e é mais fácil de realizar consultas (MONGODB, 2021). Com isso o MongoDB torna o projeto mais ágil e escalável, possibilitando executar mudanças em seus esquemas com mais velocidade à medida que a aplicação muda ou evolui.

4.6 API REST

API (RED HAT, 2021) corresponde ao acrônimo *Application Programming Interface* (Interface de Programação de Aplicações) e funciona como uma forma de integração entre aplicações distintas. Esta integração é feita a partir de normas que possibilitam a comunicação através de uma série de padrões e protocolos. Por exemplo, para um aplicativo mobile se comunicar com a câmera do celular, o mesmo poderá ter acesso a ela através da API do sistema operacional presente no celular.

Ligado ao desenvolvimento de aplicações Web, o REST, que corresponde a *Representational State Transfer* (Transferência de Estado Representacional), é estilo de arquitetura para desenvolvimento de *web services* e API's. Ele utiliza o protocolo HTTP como comunicação padrão por utilizar uma interface de operações padronizadas. E as principais vantagens de utilizar uma API REST é a separação entre cliente e servidor, mais escalabilidade, visibilidade e confiabilidade.

4.7 SPA

O conceito de *Single Page Application* (SPA) (CAVALCANTE, 2018) é aplicações Web que consistem em uma única página. Isso quer dizer que todas as funcionalidades da aplicação são centralizadas em um só lugar. Com isso, não é necessário redirecionar o usuário a uma nova página ou recarregar uma página por inteiro. Apenas o conteúdo essencial é atualizado de forma assíncrona, enquanto toda a estrutura da página permanece estática.

4.8 PROTOCOLOS HTTP E HTTPS

HTTP (GOURLEY, 2002) corresponde as palavras *Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto) é o protocolo base para a comunicação do tipo cliente-servidor na internet. Essa comunicação é feita através de requisições e respostas. Por exemplo, o cliente faz uma requisição ao servidor, o servidor retorna uma resposta contendo informações do estado da requisição, junto com o conteúdo solicitado pelo cliente no corpo de sua mensagem. Já o HTTPS significa *hypertext transfer protocol secure* (Protocolo de

Transferência de Hipertexto seguro) é a versão criptografada do HTTP. É usado para fazer esta comunicação de forma segura.

4.9 FRAMEWORKS E BIBLIOTECAS

O projeto utiliza algumas bibliotecas de apoio, tanto para o desenvolvimento quanto para o funcionamento do sistema, dentre as principais, estão o React (FACEBOOK, 2021), Axios (AXIOS, 2021) e Material UI (MATIRIAL-UI, 2021). Uma biblioteca nada mais é do que um conjunto de funções que um ou mais desenvolvedores criaram para facilitar uma certa tarefa, normalmente abstraindo elementos da linguagem e unindo-os em funções genéricas. Conceito muito confundido com o de framework, porém existem diferenças marcantes e essenciais entre esses dois tipos de ferramentas.

O *framework* é uma estrutura que rege o desenvolvimento do projeto. Ou seja, existem regras e conceitos que devem ser seguidos em sua utilização, serve para definir a aparência de como a arquitetura do projeto será. É uma ferramenta muito útil, já que abstrai dificuldades e diminui a verbosidade em trabalhar com determinada linguagem. Porém o *framework* pode acabar engessando o projeto, sendo um caminho sem volta quando o projeto já está muito grande para uma migração.

A grande semelhança entre *framework* e biblioteca, é a abstração que facilita o desenvolvimento, muitas vezes funcionalidades que seriam feitas em 10 a 15 linhas de código, podem ser feitas em 1-2 linhas com o auxílio dessas ferramentas.

Como citado acima o projeto utiliza diversas bibliotecas e *frameworks*, mas as principais são:

1. Bibliotecas e *frameworks* de *front-end*:
    a. React: é uma biblioteca JavaScript/Typescript de código aberto com foco em criar interfaces de usuário (*front-end*) em páginas web. É mantido pelo Facebook, Instagram e outras empresas, além de contar com uma comunidade de desenvolvedores individuais.

b. Axios: é uma biblioteca de cliente HTTP baseado em promessas. Utilizado para fazer toda a parte de requisições e configurações tanto na parte do cliente, quanto na parte de servidor.
c. Ant Design (XTECH, 2021): biblioteca de componentes gráficos. Utilizada para facilitar a criação das páginas, provendo estilos e componentes prontos, com classes e propriedades de fácil utilização.
d. Styled-components (JOHNSON et al, 2021): biblioteca para React que permite escrever código CSS dentro do Javascript. Isso significa que é possível usar estilos em nível de componente. Ela foi escolhida por trazer praticidade na hora de criar estilos.
e. Redux (ABRAMOV et al, 2019): biblioteca utilizada para gerenciar os estados de aplicações JavaScript. Com ela é possível centralizar o armazenamento dos estados, podendo utilizar todos os componentes ao mesmo tempo de forma compartilhada. Com isso é possível controlar estado global da aplicação.
f. MomentJs (MOMENT.JS, 2021): biblioteca JavaScript para manipular data e tempo na aplicação.

2. Bibiotecas e *frameworks* de *back-end*:

a. Mongoose (KARNIK, 2018): biblioteca NodeJs que facilita a modelagem de dados da aplicação proporcionada por uma solução baseada em esquemas. Também mapeia objetos do MongoDB e os traduz para objetos JavaScript, fazendo que eles possam ser utilizados pela aplicação de forma mais acessível.
b. Express (OPENJS, 2021): *framework* que em conjunto com o NodeJs facilitam o desenvolvimento de aplicações. Simplifica o sistema de rotas, possibilita o tratamento de exceções dentro da própria aplicação e gerencia requisições HTTP. Além de tudo, permite a integração com sistemas de templates para facilitar o desenvolvimento de aplicações Web.

5 ESTADO DA ARTE

Com o auxílio da pesquisa de mercado realizada, foram utilizados para este estado da arte os quatro softwares mais respondidos pelos entrevistados, ambos têm propósitos semelhantes ao software implementado. Com isso, estes sistemas foram avaliados e testados para que se consiga comparar as suas funcionalidades com as funcionalidades do SCOUT. Os softwares escolhidos foram:

5.1 TRELLO

O Trello é uma ferramenta de gerência de projetos que pode ser usada tanto para o acompanhamento de tarefas pessoais quanto para organizar projetos que envolvem equipes numerosas em grandes empresas. Nela o usuário pode criar quadros no qual são inseridas listas e dentro destas listas podem ser criados cartões com descrição, *upload* de arquivos multimídia, objetivos, entre outras coisas. Estes quadros podem ser compartilhados com qualquer pessoa que possui uma conta no Trello, com isso é possível atribuir usuários a cartões determinando que o mesmo é responsável por esta tarefa.

5.2 KAIRÓS

O sistema Kairós (DIMEP, 2021) segue a linha de um ponto eletrônico, onde o usuário registra os horários de entrada e saída, tanto para o início e final de expediente, quanto para os momentos de intervalo. Além do registro de horas, é possível determinar o tipo de cada jornada, ou seja, se foi realizada no ambiente de trabalho, alocado em algum cliente, alocado em *cowork*, alocado em *homeoffice*, entre outras opções. Os horários registrados tem que passar por uma aprovação de um usuário com permissões superiores, pode ser um gerente ou até mesmo supervisor da área. Após a aprovação, os horários registrados ficam à disposição de determinados usuários para consulta e geração de relatórios.

5.3 JIRA

A plataforma Jira ajuda as equipes a planejar, atribuir, acompanhar, criar relatórios e gerenciar o trabalho, unindo as equipes em tudo, desde o desenvolvimento de software e suporte ao cliente com agilidade. O Jira pode ser usado com a metodologia Scrum, Kanban ou os dois juntos de uma forma mista. Fornece estimativas que ajudam a equipe a se tornar mais precisa e eficiente de uma forma transparente para que toda a equipe se mantenha a par de tudo.

5.4 ASANA

A Asana (ASANA, 2021) é uma plataforma de gestão de projetos e equipes onde é possível criar projetos com um conjunto de tarefas a serem executadas que podem ou não seguir uma ordem pré-definida. As tarefas e fluxos de atividades que podem ser compartilhados com membros do time e nelas podem ser definidos prazos e responsáveis. Segue a metodologia ágil kanban (RADIGAN, 2021), podendo passar a atividades de uma coluna para outra conforme vão sendo finalizadas. Além da visualização por colunas, se pode usar a forma de lista de tarefas, cronograma ou calendário.

5.5 COMPARAÇÃO

A Tabela 1 abaixo compara os sistemas acima introduzidos com o sistema SCOUT. Percebe-se que somente o SCOUT oferece simultaneamente o controle de *tasks* e ponto eletrônico. Nenhum dos sistemas da tabela oferece todas funcionalidades.

Tabela 1 - Comparação entre SCOUT e os demais softwares

Funcionalidade	SCOUT	Trello	Kairos	Jira	Asana
Criação de task	X	X		X	X
Geração de Relatórios	X			X	X
Ponto eletrônico	X		X		
Suporte a plugins		X		X	X
Chat colaborativo					X

6 RESULTADOS

Neste capítulo serão apresentados os resultados de acordo com a metodologia.

6.1 DEFINIÇÃO DOS OBJETIVOS DA SOLUÇÃO

6.1.1 Identificação dos objetivos

Os objetivos da solução foram elencados no início do trabalho e estão apresentados na Seção 3.1.

6.1.2 Pesquisa de Mercado

De forma a dar completude à definição dos objetivos, realizou-se uma pesquisa de mercado em forma de formulário, neste caso a ferramenta utilizada foi o Google Forms, logo após que a ideia do projeto foi consolidada. Nela foi escrito um breve resumo do que seria o software e suas funcionalidades. O formulário foi aberto ao público dia 27/04/2020 e foi fechado dia 05/05/2020, o mesmo foi publicado nas redes sociais dos membros do projeto e enviados em grupos de aplicativos de mensagem. As perguntas foram:

a. Qual sua área de formação? (Obrigatória)
b. Caso necessário, especifique a área:
c. Atua na área de formação? (Obrigatória)
d. Você utiliza algum software parecido? (Obrigatória)
e. Caso sim, qual software?
f. Haveria espaço para o nosso software em sua empresa?
g. Utilizaria nosso software para projetos pessoais ou até mesmo para tarefas rotineiras? (Obrigatória)
h. Alguma sugestão de funcionalidade para o software, tanto em âmbito profissional como pessoal?

O formulário totalizou 132 respostas contendo dados dos entrevistados e opiniões sobre funcionalidades que os mesmos gostariam de ver no software que lhes foi apresentado. As respostas podem ser encontradas no Anexo A.

Após a análise das respostas foi aplicado um filtro na pergunta “Caso sim, qual software?”, onde foram extraídos o conjunto de respostas de cada entrevistado que respondeu entre um dos 4 softwares mais relevantes, este filtro foi aplicado por motivos de limitação de tempo. Os softwares em ordem decrescente foram: Jira (19 respostas), Trello (7 respostas), Kairos (3 respostas) e Asana (3 respostas). E dentro do conjunto de respostas destes 32 usuários coletamos todas as respostas não nulas para a pergunta “Alguma sugestão de funcionalidade para o software, tanto em âmbito profissional como pessoal?” e as respostas foram:

a. Descrição de demandas com prazo de entrega e responsável.
b. Controle de entregas da equipe.
c. Implementar um processo estruturado de UX e UI para que se torne bem atrativo."
d. Quanto mais customizável melhor!
e. Algo que misture a simplicidade do Trello mas com alguns recursos do Jira.
f. Ele deve ter um workflow visível e organizado.
g. Opção para fazer chamada de vídeo (exemplo: zoom) podendo visualizar as informações inseridas na plataforma.
h. Tem que ter sincronização com o calendário do google!
i. Por ser financeiro gostamos de ver o faturamento de cada projeto.
j. Procuro software mais simples e funcional.

Foi feita a análise das respostas para saber quais delas se aplicam aos propósitos do software, quais delas podem ser implementadas em trabalho futuros e quais não seriam usadas.

As respostas c, d, e, f, j foram rejeitadas pela equipe pelo fato de serem muito subjetivas, pois cada usuário pode ter a sua visão de algumas características citadas como simples, funcional, visível e organizado. As respostas g, h, i poderão ser implementadas em trabalhos futuros por serem melhorias específicas de determinadas funções e fugirem do escopo do software proposto neste documento. As respostas a, b, foram as que mais se encaixaram na finalidade do software e por isso foram implementadas.

Outra resposta relevante foi que 86% dos entrevistados utilizariam o software para projetos pessoais ou tarefas rotineiras. Isso leva a pensar que o problema levantado acima afeta muitas pessoas e com a solução proposta, esta adversidade pode ser amenizada.

6.2 PROJETO E DESENVOLVIMENTO DA SOLUÇÃO

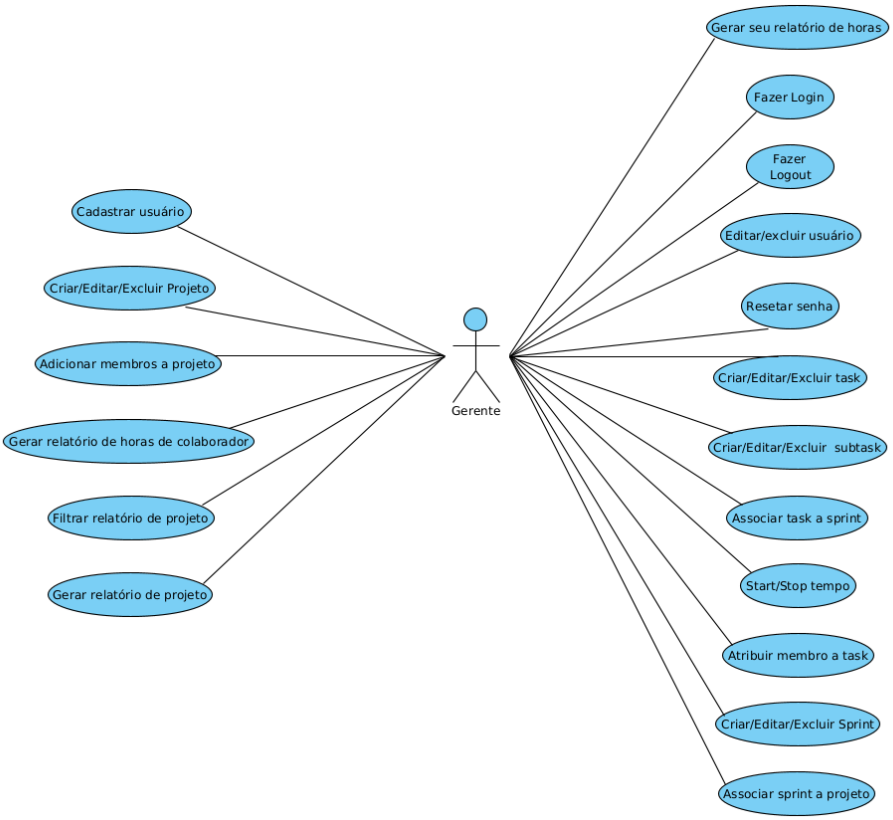
6.2.1 Casos de uso

Os diagramas de caso de uso em UML (*Unified Modeling Language*) (DRUSINSKY, 2011) servem para mostrar as maneiras em que o usuário, independentemente da sua função, interage com o sistema. Nele são apresentados os tipos de usuário e a funcionalidade que cada um pode executar no sistema.

Nos diagramas a seguir, é mostrado, de maneira simplificada, todas as principais funcionalidades do SCOUT atribuídas aos dois tipos de usuários que o mesmo possui: Gerente, Figura 2 e Colaborador, Figura 3.

Casos de uso para o ator Gerente:

Figura 2 – Casos de Uso Gerente



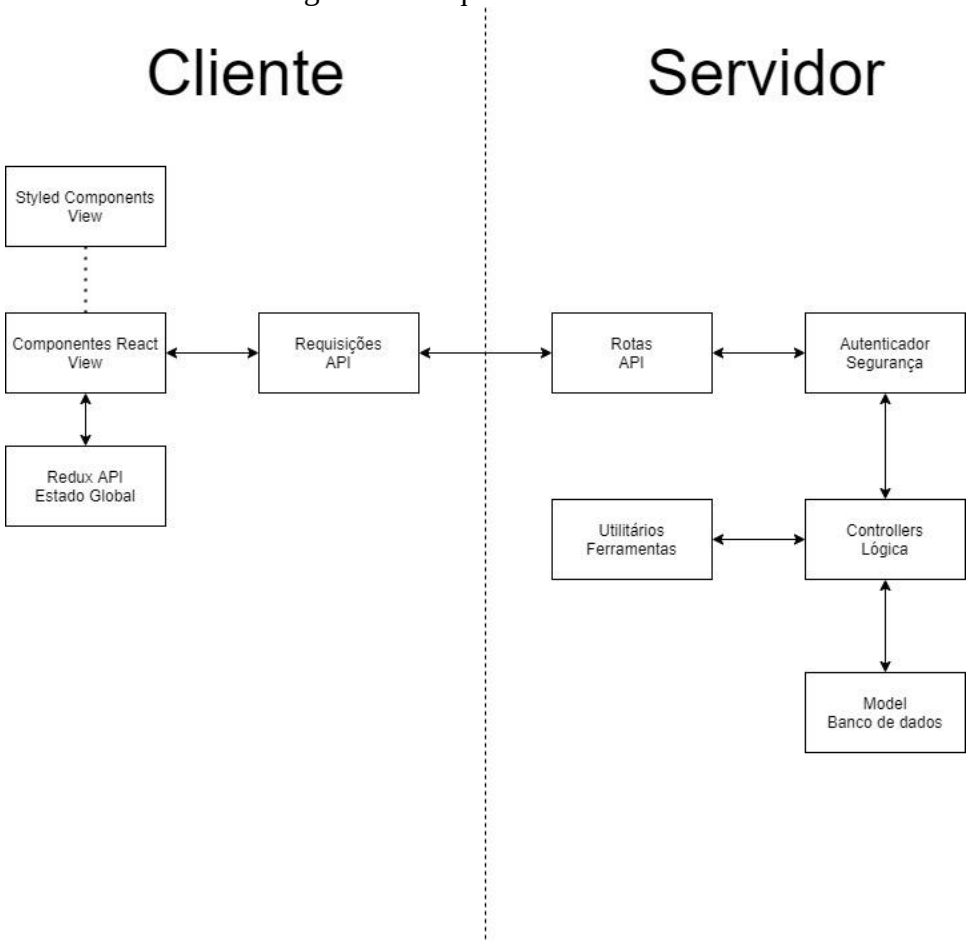
Casos de uso para o ator Colaborador:

Figura 3 – Casos de uso Colaborador

6.2.2 Visão Geral da Arquitetura do Sistema

A arquitetura utilizada no SCOUT segue o modelo Cliente-Servidor e está representada na Figura 4 abaixo. Logo a seguir, será explicado cada um de seus componentes e suas funcionalidades.

Figura 4 – Arquitetura do Sistema



6.2.2.1 Cliente

Components React View:

É composto por componentes *React* no formato .tsx, que é a união de *templates* HTML com lógica em *Javascript.* A ideia é que cada componente possua um estado interno, tendo sua lógica com escopo adaptado às necessidades do mesmo. Toda estrutura de telas é formada por componentes, que vão se dividindo em componentes menores até se tornarem "átomos" que possuem funções bem determinadas e de fácil manutenção. Existem dois tipos de componentes envolvidos nessa arquitetura. Dois deles são diferenciados apenas por sua origem, um surge do próprio desenvolvimento da equipe e outro vem da biblioteca chamada *AntDesign.* Ambos possuem lógica própria e são usados para criação de alto nível da aplicação.

Styled Components View:

Componente usado apenas para estrutura mais baixa, onde a lógica embutida nele serve apenas para alterar o estilo. Esses componentes são criados a partir da biblioteca *Styled Components* e têm a função de criar os estilos da interface do projeto.

Redux API Estado Global:

Para o controle de estado global da aplicação foi utilizado o *Redux* com sua nova versão de API, já integrada com os *hooks* do *React*. O estado global serve para compartilhar informações e determinar estados entre os componentes. Normalmente os componentes se comunicam apenas de pai para filho e vice-versa, para isso existe essa estrutura, para que componentes irmãos ou até mesmo componentes sem ligação possam se comunicar e compartilhar estados. Um exemplo simples disso é quando um componente executa uma chamada no servidor do projeto (*back-end*) e determina que a aplicação do usuário deve entrar em estado de carregamento. Assim, todos os outros componentes da aplicação sabem que essa ação está ocorrendo e que não podem executar outras ações.

Requisições e API:

O tipo de requisição escolhido para o projeto foi o REST, que é gerenciada a partir da ferramenta Axios. Essa parte consiste em funções integradas com o Axios, que podem ser utilizadas por qualquer componente na aplicação para buscar ou alterar dados na parte de servidor. Cada função tem os parâmetros determinados de acordo com a chamada REST.

6.2.2.2 Servidor

Controllers e Lógica:

Toda a lógica bruta da aplicação está reunida nos *controllers*, que são arquivos JavaScript contendo as funções necessárias para cada ação. Essas funções estão agrupadas em módulos do NodeJS, a tecnologia usada como base para o servidor. Caso uma requisição esteja autorizada, a mesma irá para o *controller* responsável e nesse momento haverá o acesso ao banco de dados por meio dos models, caso necessário. Quando a ação for completada, o *controller* enviará por *callback* a resposta para o cliente contendo o sucesso ou o erro dessa ação.

API e Rotas:

As rotas são responsáveis pela chegada das requisições vindas do cliente e aqui são determinados os "*endpoints*", que são as representações textuais finais de cada link. Um exemplo básico seria de uma rota de login, onde as rotas determinam a parte final como "/login" e o link formado na parte do cliente deveria ser "URL do endereço base do servidor" + "/login". Nas rotas também são determinadas quais devem passar pelo sistema de autenticação e quais devem ser direcionadas diretamente para os *controllers*. As rotas são criadas de acordo com as especificações da ferramenta de serviço utilizada, nesse caso o Express.

Autenticador e Segurança:

A segurança é feita por uma entidade separada dos *controllers*, que determina caso a requisição e o usuário que a fez estão autorizados a executar tal ação. Um exemplo simples seria o usuário tentando alterar as informações de uma *task* sem estar autenticado no sistema, o que deveria retornar um erro, impedindo que a ação chegasse nos *controllers*. A segurança dessa aplicação é feita baseada na chave que o usuário envia no cabeçalho da requisição, essa chave é verificada pela ferramenta JWT, que gerencia a criação e verificação dessas chaves nas sessões da aplicação. Vale lembrar que essas chaves são geradas e enviadas para o cliente no momento do *login*.

Utilitários e Ferramentas:

Nesta parte ficam todas as ferramentas extras necessárias para o funcionamento da aplicação. Tudo que não está diretamente ligada a lógica dos *controllers* porém deve ser utilizado em certos momentos. Um exemplo básico seria a ferramenta de e-mail, que é utilizada na ação de recuperação de senha e que consiste em uma ferramenta criada a partir de uma biblioteca externa, a NodeMailer. Dentro dessa seção estão também os adaptadores de dados ou até mesmo os tradutores necessários, que são responsáveis por toda a manipulação/conversão de estrutura de dados. Eles estão localizados aqui porque podem ser utilizados por todos os *controllers*, sem escopo definido.

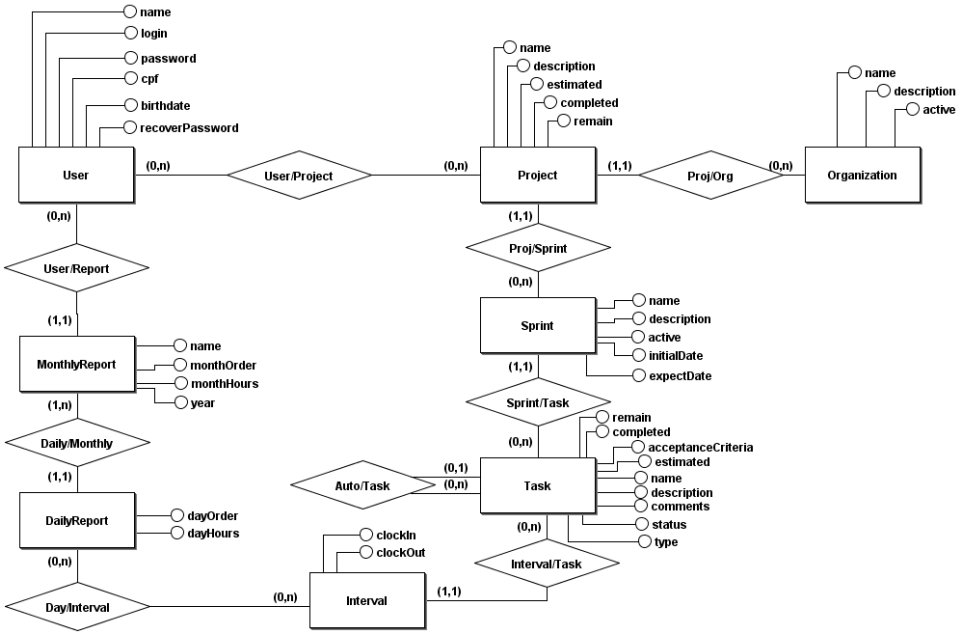
Model e Banco de dados:

O banco de dados escolhido foi o MongoDB, banco orientado a documentos, ou seja, possui uma linguagem de requisições própria, sendo assim é um banco NoSQL. O banco é formado a partir dos *Models*, que são as representações em código das entidades presentes. Esses *Models* são criados a partir da junção do MongoDB com a ferramenta Mongoose, que é um gerenciador de requisições e tradução de dados. Todas as requisições partem dos *controllers*, porém devem passar pelos Models e respeitar a sintaxe utilizada pelo Mongoose. O Mongoose encapsula a lógica do banco em funções com parâmetros definidos de acordo com cada uma das ações que o *controller* deseja realizar.

6.2.3 Modelo Entidade Relacionamento

Para se entender melhor os objetos presentes no SCOUT, seus atributos e como eles se relacionam, foi criado um Diagrama de Entidade Relacionamento apresentado abaixo na Figura 5.

Figura 5 – Diagrama Entidade Relacionamento

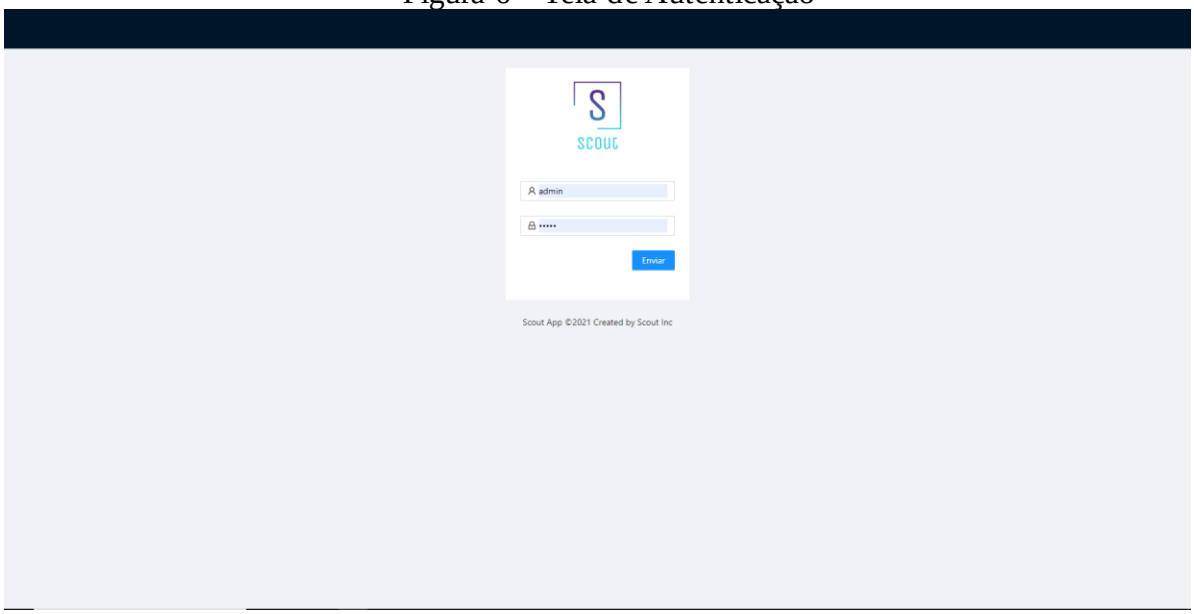


6.2.4 Principais Telas do Sistema

Nesta sessão serão apresentadas e detalhadas as principais telas do sistema.

A tela de Autenticação é composta pelos campos de usuário e senha, como mostrado na Figura 6.

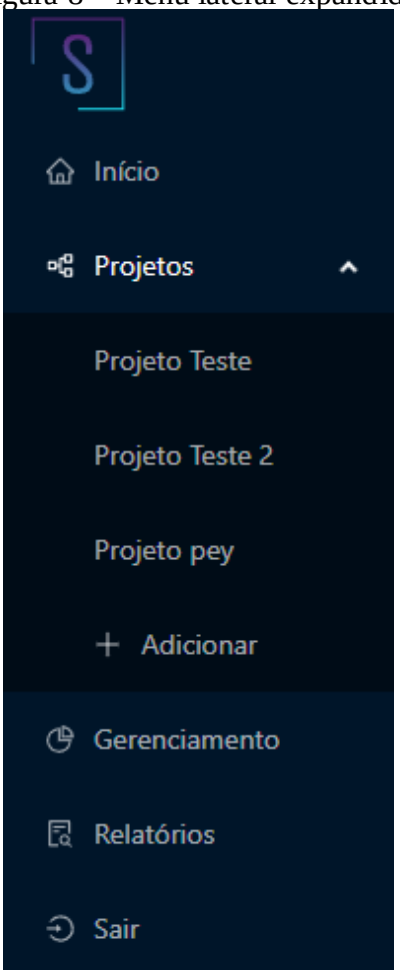
Figura 6 – Tela de Autenticação



A tela Inicial/Home é composta pela data, botão para iniciar a função de Ponto Eletrônico, botão para pausar a função de Ponto Eletrônico. Os últimos lançamentos do usuário, que são os horários que o usuário trabalhou em um determinado dia e as *tasks* que o mesmo executou. E por fim, uma lista das *tasks* atribuídas ao usuário. Esta estrutura é mostrada na Figura 7. Também na tela Inicial/Home é encontrado um menu lateral que leva o usuário para as demais telas, como mostrado de modo expandido na Figura 8.

Figura 7 – Tela Inicial/Home

Figura 8 – Menu lateral expandido



A tela de Projeto, mostrada na Figura 9, é composta pelas informações do projeto, contendo data de criação, usuários do projeto, identificando quem é o administrador e uma lista de *tasks* vinculadas ao projeto. Ela é composta também do botão "Editar Projeto", no qual abre-se um modal em que o usuário pode alterar as informações do projeto como mostrado na Figura 10. O botão de "Gerenciar membros do projeto", habilitado apenas para o administrador, no qual abre-se um *modal* mostrado na Figura 11 e o botão de "Adicionar *Task"*, no qual abre-se um *modal* mostrado na Figura 12.

Figura 9 – Tela de Projeto

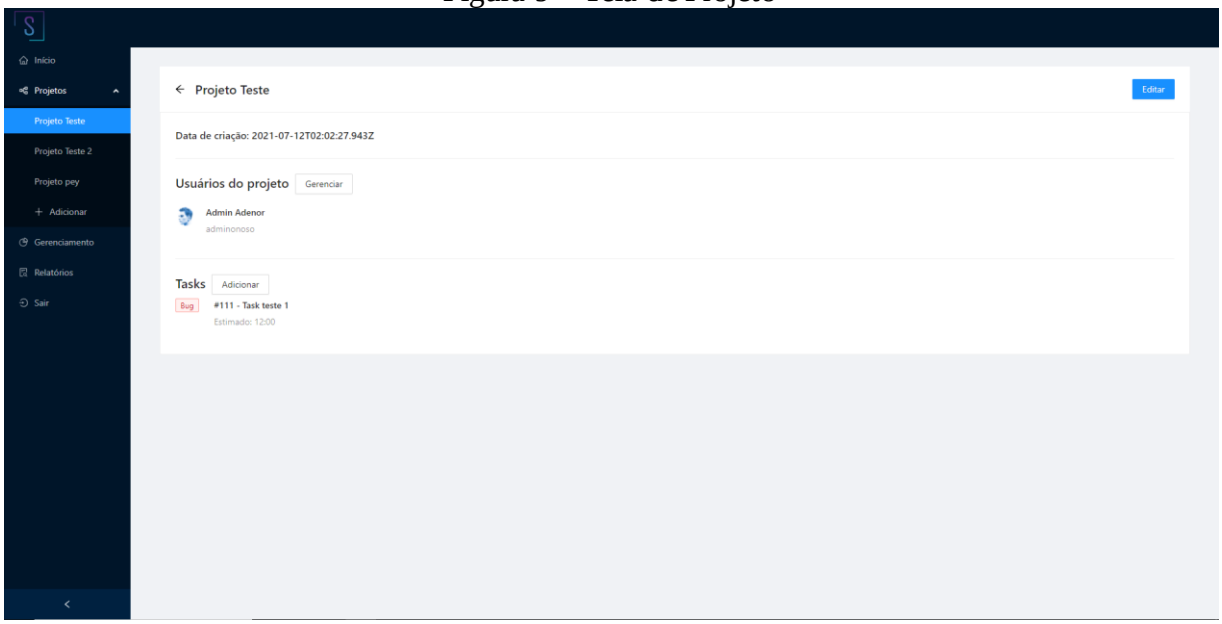


Figura 10 – *Modal* Editar Projeto

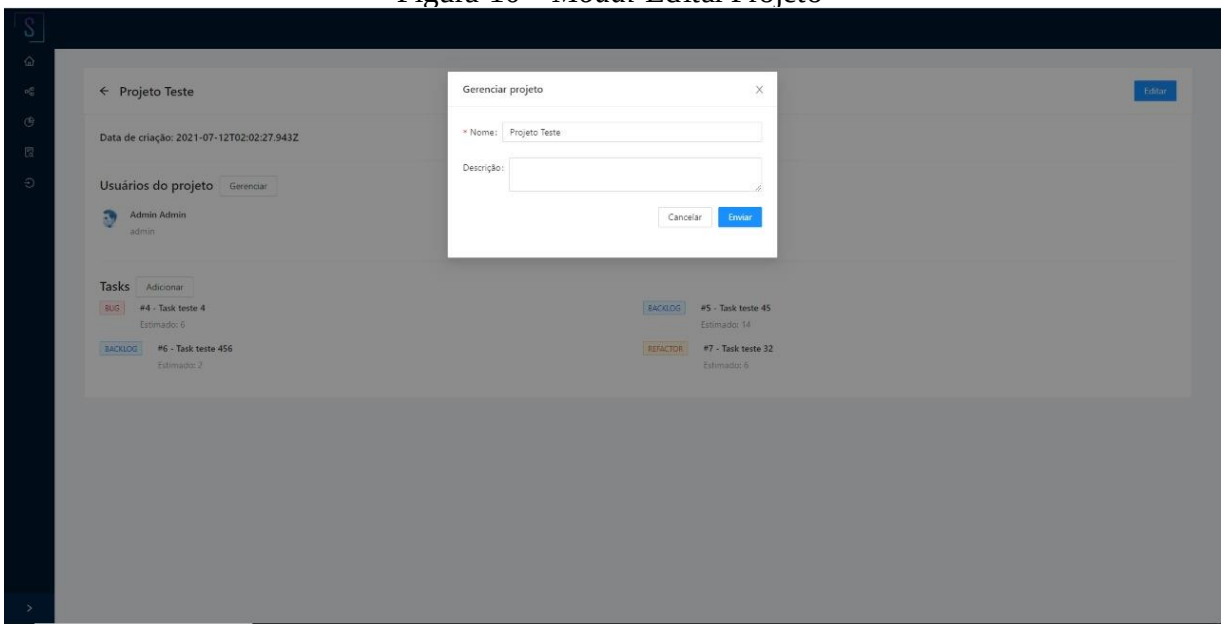


Figura 11 – *Modal* de Gerenciar Membros do Projeto

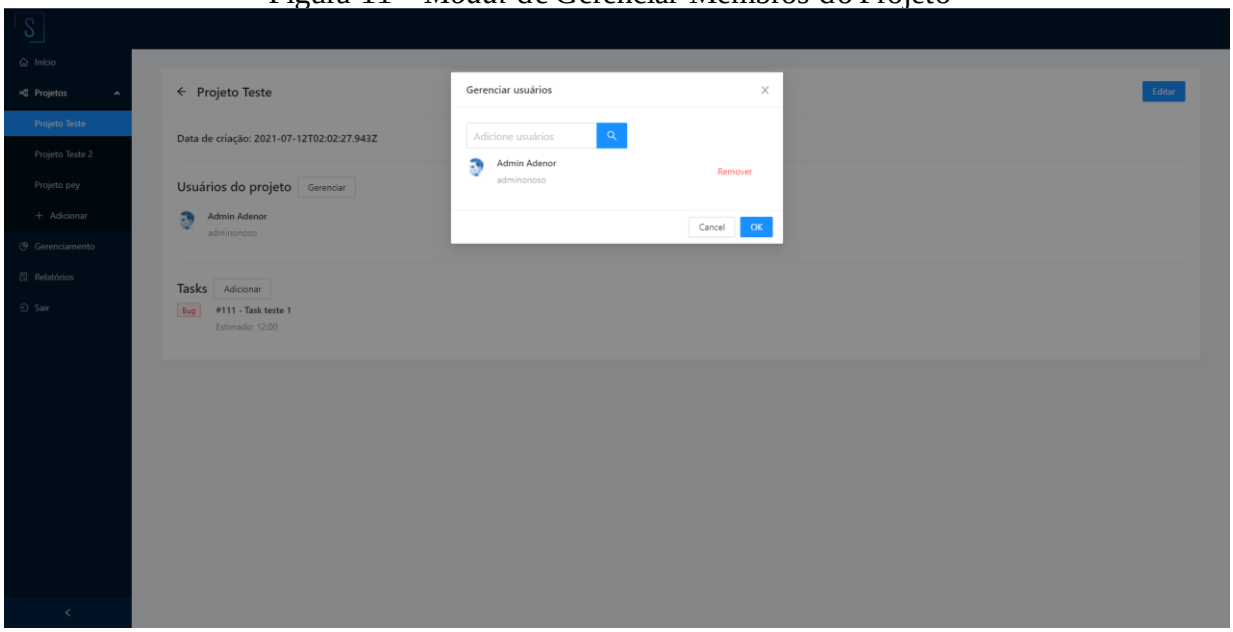
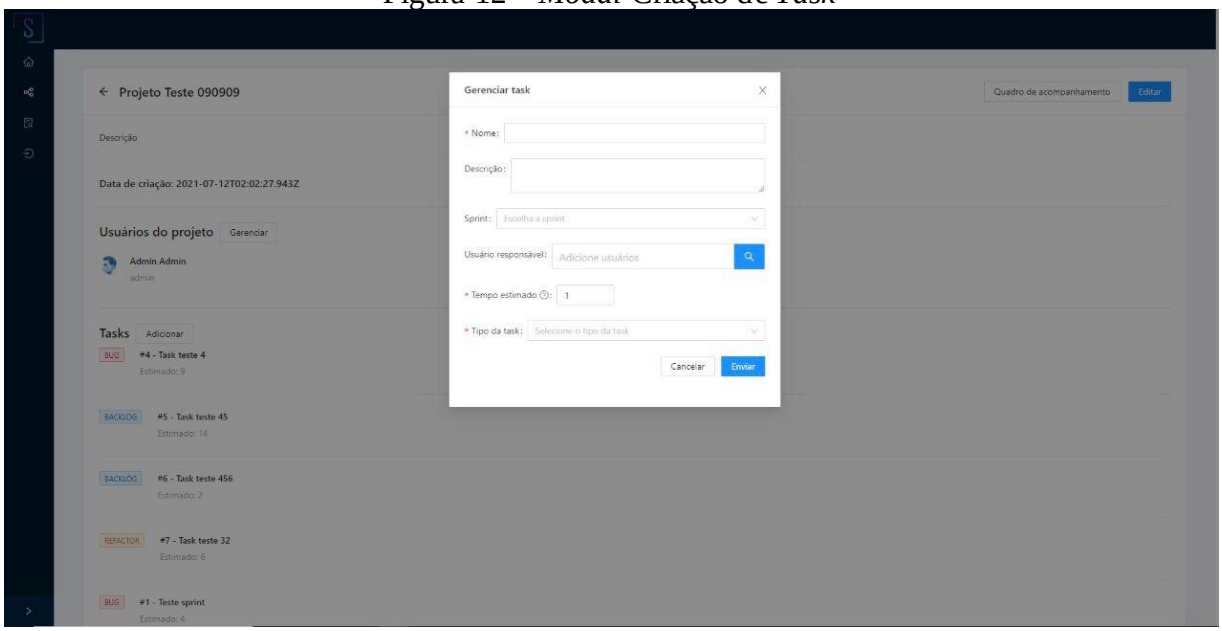
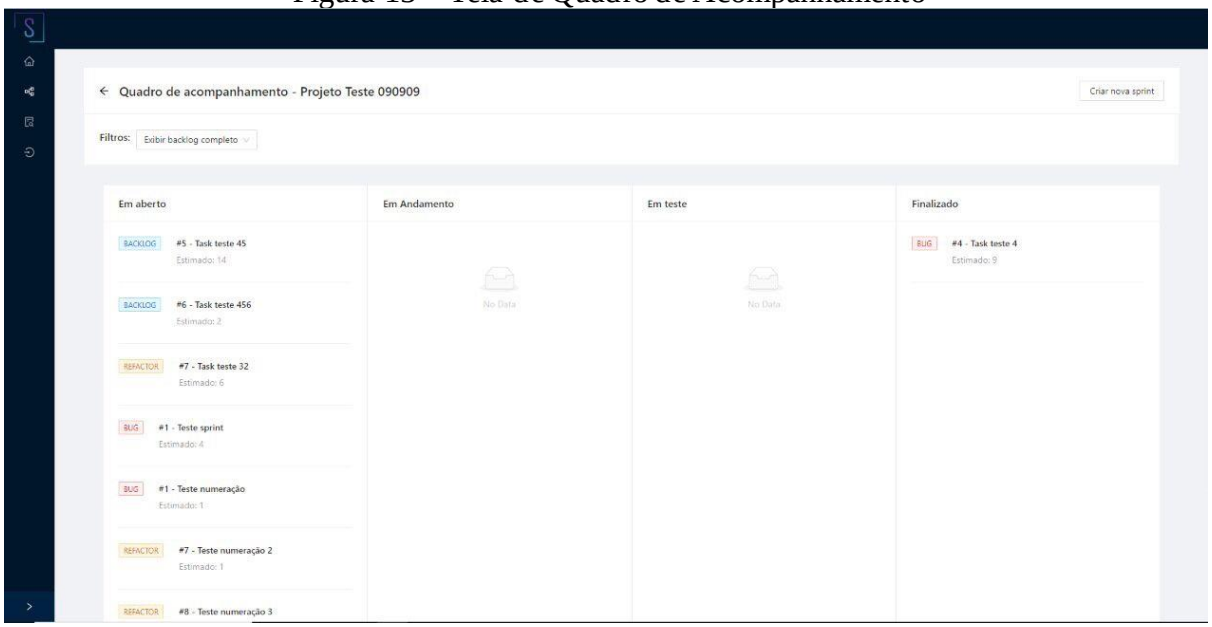


Figura 12 – *Modal* Criação de *Task*



A tela de Quadro de Acompanhamento é composta de um quadro visual de um determinado projeto com as colunas “Em Aberto”, “Executando”, “Testando” e “Finalizado”. Onde as *tasks* são organizadas automaticamente dependendo do estado de cada uma como mostrado na Figura 13.

Figura 13 – Tela de Quadro de Acompanhamento



A tela de Relatório de Horas, mostrada na Figura 14, permite visualizar as horas trabalhadas pelo usuário no mês que o usuário filtra. A tela é composta de um calendário onde cada dia é possível selecionar as horas trabalhadas. Ao selecionar estas horas, é aberto um modal onde se pode visualizar o intervalo de horas trabalhadas naquele dia e as *tasks* que o usuário estava trabalhando naquele intervalo de horas, como mostrado na Figura 15. E também, ao selecionar um dia do calendário, é aberto o modal de "Informações Diárias". Esse cadastro é feito manualmente. Ao selecionar o botão "Adicionar novo intervalo", o sistema preenche o campo de hora inicial automaticamente com o horário real, sendo que o usuário pode alterá-lo. O horário final fica em branco, mas o usuário tem a opção de preenchê-lo, caso ele já tenha feito estas horas anteriormente. O modal de "Informações Diárias" é mostrado na Figura 16.

Figura 14 – Tela de Relatório de Horas

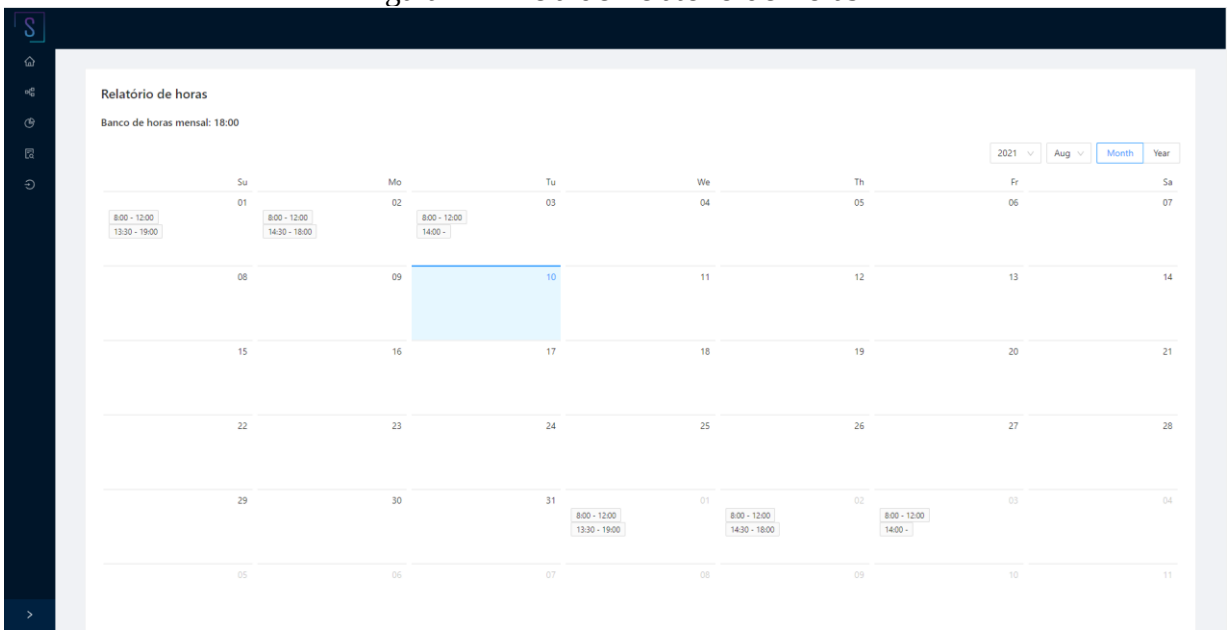


Figura 15 – *Modal* Visualizar Horas

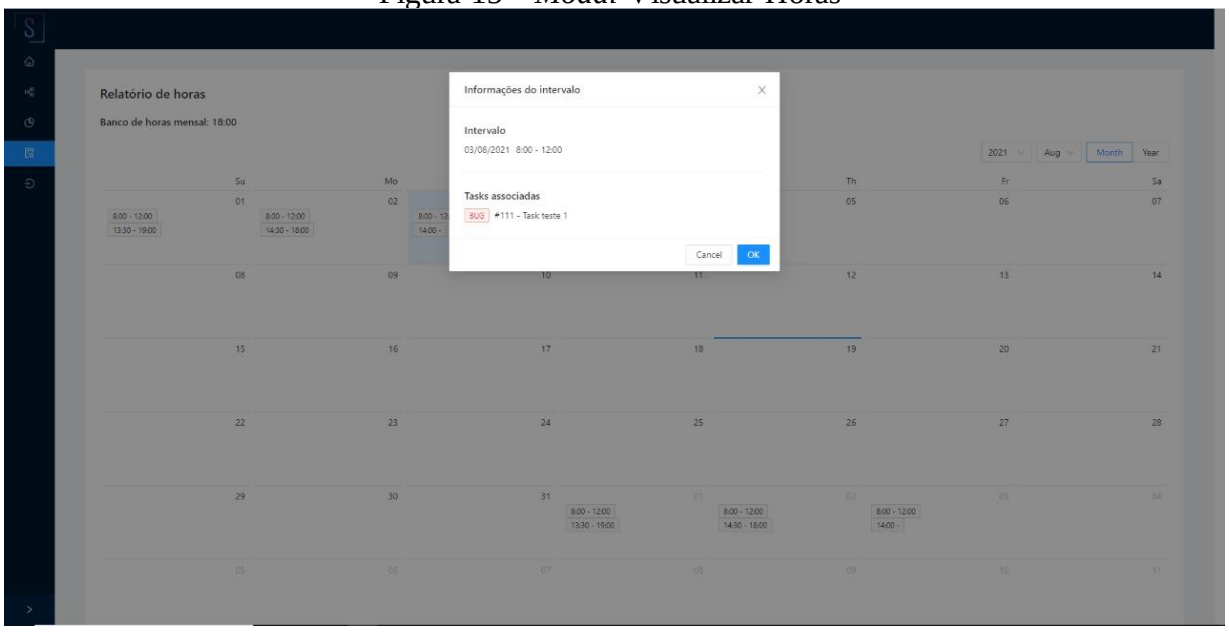
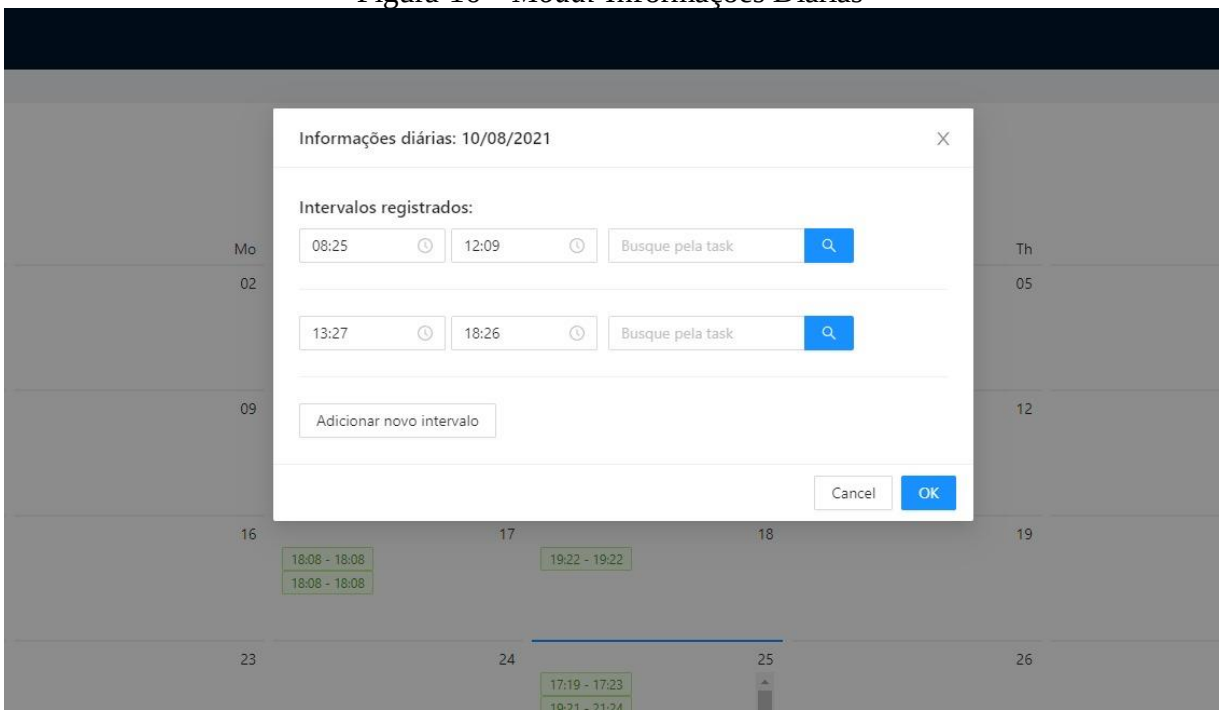


Figura 16 – *Modal* Informações Diárias



7 AVALIAÇÃO

7.1TESTES FUNCIONAIS

Para verificar que o SCOUT cumpre todos os requisitos impostos a ele com o menor número de erros possíveis, foram realizados testes funcionais. Os mesmos são executados analisando as saídas do sistema, perante entradas pré-determinas. Também são conhecidos como teste de caixa-preta por não terem contato com o código, mas somente com o programa ou parte dele (WAZLAWICK, 2013). Foi testada cada parte do sistema individualmente e depois todas as partes juntas como um todo. Com isso, foi capaz de medir a qualidade e a confiabilidade da aplicação.

Para auxiliar na organização da execução destes testes, foram criados uma série de casos de teste. Eles servem para documentar as entradas, o que acontece e as saídas esperadas de uma determinada funcionalidade do sistema. E o seu principal objetivo é encontrar erros. Abaixo, está sendo mostrado alguns dos principais casos de teste:

ID: 4

Caso de Teste: Validar se hora está sendo registrada com sucesso.

Tela: Relatório de Horas.

Pré-requisito: Estar logado no sistema com qualquer tipo de usuário.

Passos de teste:

1 – Logar no sistema com qualquer tipo de usuário.

2 – Acessar tela de "Relatório de Horas".

3 – Selecionar dia que o usuário quer registrar a hora.

4 – No modal de Informações diárias, selecionar o botão "Adicionar novo intervalo".

5 – Preencher as horas e selecionar o botão "Ok"

Resultados esperados: Sistema emite toast com a mensagem "Horas registradas com sucesso" e a hora é mostrada no dia selecionado pelo usuário no Passo 3.

Resultado atual: Sistema emite toast com a mensagem "Horas registradas com sucesso" e a hora é mostrada no dia selecionado pelo usuário no Passo 3.

Status: OK

ID: 7

Caso de Teste: Validar se projeto está sendo adicionado com sucesso.

Tela: Adicionar Projeto.

Pré-requisito: Estar logado no sistema com um usuário com perfil Administrador.

Passos de teste:

1 – Logar no sistema com um usuário com perfil Administrador.

2 – Acessar tela de "Adicionar projeto".

3 – Preencher campos obrigatórios.

4 – Selecionar o botão "Adicionar".

Resultados esperados: Sistema emite toast com a mensagem "Projeto adicionado com sucesso" e é mostrado a tela do projeto criado.

Resultado atual: Sistema emite toast com a mensagem "Projeto adicionado com sucesso" e é mostrado a tela do projeto criado.

Status: OK

ID: 19

Caso de Teste: Validar se status da task muda ao gerenciar task.

Tela: Modal Gerenciar Task.

Pré-Requisitos: Estar logado no sistema com um usuário que seja membro do projeto que a task faz parte.

Passos de teste:

1 - Logar no sistema com um usuário que seja membro do projeto que a task faz parte.

2 – Selecionar projeto no qual a task faz parte.

3 – Selecionar a task que se deseja mudar o status.

4 – Selecionar o botão “Gerenciar task”.

5 – Alterar Status da task.

6 – Selecionar o botão “Enviar”.

Resultado esperado: Sistema emite toast com a mensagem "Task atualizada com sucesso" e status da task é alterado.

Resultado atual: Sistema emite toast com a mensagem "Task atualizada com sucesso" e status da task não é alterado.

Status: Não OK.

Alguns erros foram encontrados, como o do caso de teste id 19 acima. Os mesmos foram registrados e corrigidos. Depois os testes eram refeitos até que todos os casos de teste estivessem com o status OK. Segue o registro de erro do caso de teste id 19:

Nome: Aplicação não muda status da task ao gerencia-la no modal de "Gerenciar Task".

Descrição: Aplicação não muda o status da task no banco de dados após alterar o campo Status no modal de “Gerenciar Task”.

Passos para reproduzir:

1 - Logar no sistema com um usuário que seja membro do projeto que a task faz parte.

2 – Selecionar projeto no qual a task faz parte.

3 – Selecionar a task que se deseja mudar o status.

4 – Selecionar o botão “Gerenciar task”.

5 – Alterar Status da task.

6 – Selecionar o botão “Enviar”.

Evidencias: Após a emissão do toast com a mensagem "Task atualizada com sucesso", o status task não é atualizada no banco de dados.

Critérios de aceitação: Após a emissão do toast com a mensagem "Task atualizada com sucesso", o status task precisa ser atualizado no banco de dados e na interface.

Após finalizar os testes funcionais, o desenvolvimento do SCOUT foi concluído.

8 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Neste trabalho de conclusão de curso, foi proposto um software que une uma ferramenta de gerenciamento de projeto com as funcionalidades de um ponto eletrônico que atualiza automaticamente as informações nos projetos dentro do sistema e gera relatórios de horas, para melhor controle da jornada de trabalho. O sistema foi nomeado de SCOUT", que significa Sistema de Controle Orçamentário e Utilização de Tempo. Após a proposta ter sido aprovada, o este trabalho começou a ser planejado com a ajuda da metodologia conhecida por *Design Science Research Methodology* (DSRM). Após isso, foi realizada uma pesquisa de mercado com o objetivo de saber se este software era de interesse das pessoas. Se obteve uma resposta positiva e ainda se pôde localizar quais produtos presentes no mercado atual que mais se assemelhavam com o SCOUT. Em seguida, depois de se estudar sobre estes produtos, o desenvolvimento do sistema foi iniciado e concluído. As principais tecnologias utilizadas foram React com a linguagem de programação Javascript para o *front-end* e, Node e Express também com a linguagem Javascript para o *back-end*. Por fim, foram realizados testes funcionais no software e foi concluído que ele é funcional, viável e tem potencial para se tornar um produto.

Para trabalhos futuros, os autores destacam, realizar a avaliação do artefato, como proposto na quinta etapa da metodologia. Para isso, será necessário a criação e a aplicação de um questionário para que atores humanos avaliem se o sistema atinge os objetivos propostos na segunda etapa da metodologia. Também pode-se sugerir que o SCOUT possa sincronizar com uma conta Google para que o usuário possa importar os compromissos e reuniões da sua Google Agenda, para receber notificações via e-mail e também pode ser usada como forma de login. Outra sugestão é a criação de uma página *Analytics* para que se possa extrair os números que o SCOUT gera e de certa forma organizá-los com o objetivo de encontrar padrões e tirar conhecimento a partir deste conjunto de dados. E como última sugestão, foi pensado na integração do SCOUT com um sistema de versionamento de software, onde seria possível associar *branchs* a *tasks*. Podendo também incluir a funcionalidade de só ser possível fazer o *merge* de uma *branch* se a *task* que ela estiver associada seja finalizada, ou seja, com o status igual a finalizado.

REFERÊNCIAS

JUNIOR, V. F. et al. **Design science research methodology enquanto estratégia metodológica para a pesquisa tecnológica**. Revista Espacios, v. 38, n. 6, p. 25–34, 2017.

GOUVEA, Carlos et al. A utilização do SCRUM como recurso educacional no processo de aprendizagem em Engenharia de Software. **Revista Eletrônica Engenharia Viva**, Goiânia, v. 3, n. 2, p. 87-102, 31 mar. 2017. Semestral. Disponível em: https://www.researchgate.net/publication/325675845_A_utilizacao_do_SCRUM_como_recurso_educacional_no_processo_de_aprendizagem_em_Engenharia_de_Software. Acesso em: 22 jun. 2021.

ATLASSIAN. Jira Software - **Features**. 2021. Disponível em: https://www.atlassian.com/software/jira/features/> Acesso em: 2 jun, 2021.

ATLASSIAN. **Trello Tour**. 2021. Disponível em: https://trello.com/tour/> Acesso em: 2 jun, 2021.

MICROSOFT. **Microsoft Project**. 2021. Disponível em: https://www.microsoft.com/pt-br/microsoft-365/project/project-management-software/> Acesso em: 2 jun, 2021.

CITRIX SYSTEMS. **Tour - Citrix Podio**. 2021. Disponível em: https://podio.com/site/tour/> Acesso em: 2 jun, 2021.

SHASTRI, Y. et al. **Spearheading agile: the role of the scrum master in agile projects.** Empirical Software Engineering, v. 26, n. 3, 2021

Wazlawick, R. S. **Engenharia de Software: Conceitos e práticas**. Florianópolis: Elsevier, 2013.

Duckett, J. **HTML and CSS Design and Build Websites.** Indianapolis: Wiley, 2014.

FACEBOOK Inc. **Tutorial: Into to React**. 2021. Disponível em: https://reactjs.org/tutorial/tutorial.html/> Acesso em: 2 jun, 2021.

Haverbeke, M. **Eloquent JavaScript. A Modern Introduction to Programming.** San Francisco: no starch press, 2018

OPENJS FOUNDATION. **About**. 2021 Disponível em: https://nodejs.org/en/about/> Acesso em: 7 jun, 2021

MONGODB Inc. **What is mongodb?** 2021. Disponível em: https://www.mongodb.com/what-is-mongodb/>. Acesso em: 16 jun. 2021.

MONGODB Inc. **What is noSQL?** 2021. Disponível em: https://www.mongodb.com/nosql-explained />. Acesso em: 16 jun. 2021.

RED HAT, INC. **What is a REST API?** 2021. Disponível em: https://www.redhat.com/en/topics/api/what-is-a-rest-api/> Acesso em: 8 jun, 2021

Cavalcante, J. **Descomplicando SPA's**. 2018 Disponível em: https://medium.com/trainingcenter/descomplicando-spas-caa8f57bdbf3/> Acesso em: 8 jun, 2021

Gourley, D. et al. **HTTP: The Definitive Guide.** Sebastopol: O'Reilly, 2002

MICROSOFT. **TypeScript: Typed JavaScript at Any Scale**. 2021. Disponível em: https://www.typescriptlang.org/> Acesso em: 16 jun, 2021

THE AXIOS PROJECT. **Getting Started**. 2021. Disponível em: https://axios-http.com/docs/intro/> Acesso em: 16 jun, 2021

MATIRIAL-UI. **Aprenda Matirial-UI**. 2021. Disponível em: https://material-ui.com/pt/getting-started/learn/> Acesso em: 16 jun, 2021

MOMENT.JS. **Documentation**. 2021. Disponível em: https://momentjs.com/docs/> Acesso em: 18 jun, 2021

Karnik, N. **Introduction to Mongoose for MongoDB**. 2018. Disponível em: https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb d2a7aa593c57/> Acesso em: 18 jun, 2021

OPENJS. **Express - Node.js web application framework**. Disponível em: https://expressjs.com/> Acesso em: 18 jun, 2021

XTECH. **Introduction - Ant Design**. 2021 Disponível em: https://ant.design/docs/spec/introduce/> Acesso em: 21 jun, 2021

DIMEP. **Kairos**. 2021. Disponível em: https://www.dimep.com.br/kairos/> Acesso em: 22 jun, 2021

ASANA. **Software de gerenciamento**. 2021. Disponível em: https://asana.com/pt/uses/project-management/> Acesso em: 22 jun, 2021

Drusinsky, D. **Modeling and Verification Using UML Statecharts. A Working Guide to Reactive System Design, Runtime Monitoring and Execution-based Model Checking.** Burlington: Elsevier, 2011.

Radigan, D. **Kanban - A brief introduction**. 2021. Disponível em: https://www.atlassian.com/agile/kanban/> Acesso em: 25 jun, 2021

Barros, L. **O que fazer quando sua máquina de bater ponto é ultrapassada?** 2019. Disponível em: https://blog.tangerino.com.br/maquina-de-bater-ponto-ultrapassada/> Acesso em: 25 jun, 2021

Abramov, D. et al. **Redux Essentials, Part 1: Redux Overview and Concepts**. 2021. Disponível em: https://redux.js.org/tutorials/essentials/part-1-overview-concepts/> Acesso em: 25 jun, 2021

Johnson, A. et al. **Documentation**. 2021. Disponível em: https://styled-components.com/docs/> Acesso em: 25 jun, 2021

ANEXO A – Questionário de estudo de mercado

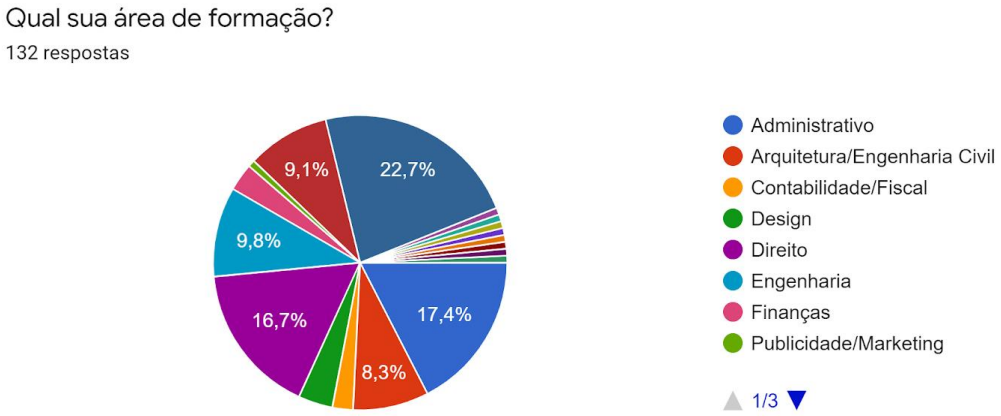
Olá! Estamos desenvolvendo um software que une as funcionalidades de um controlador de horas diretamente com um gerenciador de projetos, o SCOUT. Já existem alguns softwares parecidos bem consolidados no mercado, como o JIRA, Monday, Trello, Redmine, entre outros. A ideia é fundir a criação de equipes, projetos e tarefas com o controle de horas trabalhadas, agenda de trabalho e calendário, com uma interface simplificada e de fácil configuração, sem a necessidade de mudança de sistema. Com isso, gostaríamos de saber algumas informações para aprimorar nosso TCC e quem sabe transformar em um produto.

Obrigado desde já pela participação!

Qual sua área de formação? *

- Administrativo - 23 respostas
- Arquitetura/Engenharia Civil - 11 respostas
- Contabilidade/Fiscal - 3 respostas
- Design - 5 respostas
- Direito - 22 respostas
- Engenharia - 13 respostas
- Finanças - 4 respostas
- Publicidade/Marketing - 1 resposta
- Saúde - 12 respostas
- Tecnologia da Informação - 30 respostas
- Outra: Letras, Ciência e tecnologia de alimentos, Educação, Gestão, Comércio exterior, Estudante, Contabilidade/pessoas e Economia - todos com 1 resposta somando fica 8 respostas

RESPOSTA EM FORMA DE GRÁFICO



Caso necessário, especifique a área:

RESPOSTAS

Medicina

Comércio Exterior

Fonoaudiologia

Gerência de Projetos

Medicina Veterinária

Arquitetura de saúde e engenharia mecânica hospitalar

Design Grafico

FP&A

Design gráfico

Eletrônica

Fisioterapia em estética

Economia

Pergunta em branco

Advocacia

Engenharia Mecânica

Ciência de Dados

Ciência da Computação

Desenvolvedor Front-end

Dentista

Gerenciamento

Supply chain Management

Industrial

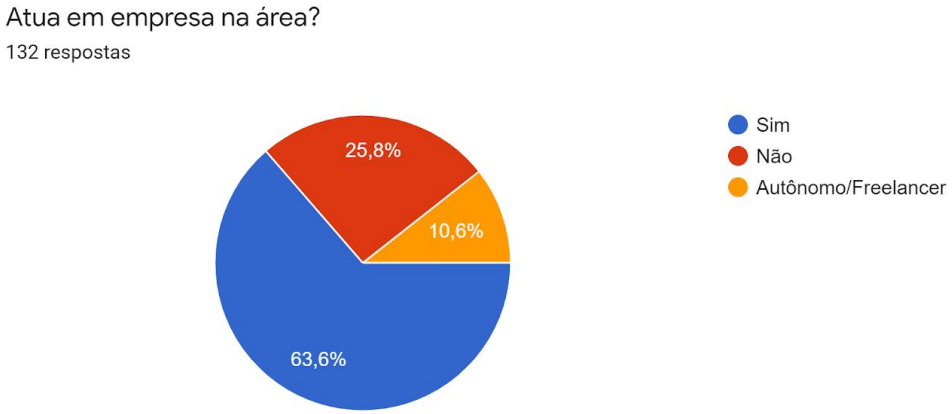
Quality Assurance

RH

Atua em empresa na área? *

- Sim - 84 respostas
- Não - 34 respostas
- Autônomo/Freelancer - 14 respostas

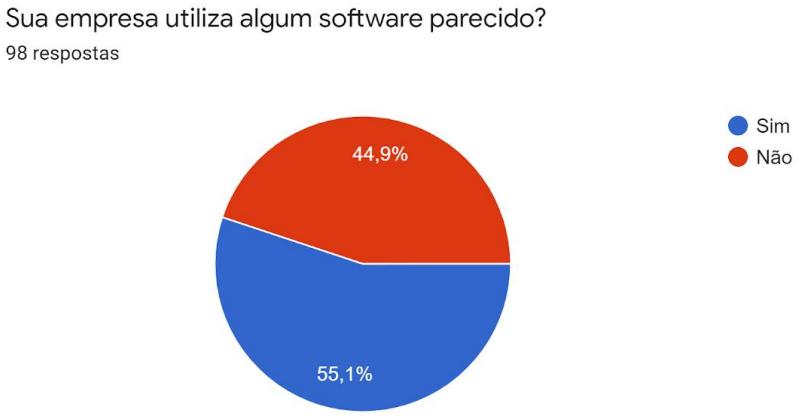
RESPOSTA EM FORMA DE GRÁFICO



Sua empresa utiliza algum software parecido? *

- Sim - 54 respostas
- Não - 44 respostas
- 34 respostas em branco.

RESPOSTA EM FORMA DE GRÁFICO



Caso sim, qual software?

RESPOSTA (Tem ocasiões com mais de uma resposta por intrevistado)

resposta em branco - 80 respostas

Jira - 19 respostas

Kairos - 3 respostas

Asana - 3 respostas

Trello - 7 resposta

Podio - 1 resposta

Domínio Sistemas - 1 resposta

Controle interno - 1 resposta

pipefy - 1 resposta

Airtable - 1 resposta

Toggl - 1 resposta

GitHub - 1 resposta

Airbox - 1 resposta

Clickup - 1 resposta

MS Project - 1 resposta

Senior sistemas/rubi - 1 resposta

Prodent - 1 resposta

Mantis - 1 resposta

Software interno - 1 resposta

agilis - 1 resposta

sap - 1 resposta

Azure - 1 resposta

T-Sheets - 1 resposta

Software da Microsoft - 3 respostas

Haveria espaço para o nosso software em sua empresa? *

- Sim - 72 respostas
- Não - 26 respostas
- em branco - 34 respostas

RESPOSTA EM FORMA DE GRÁFICO

Utilizaria nosso software para projetos pessoais ou até mesmo para tarefas rotineiras? *

- Sim - 114 respostas
- Não - 18 respostas

RESPOSTA EM FORMA DE GRÁFICO

Alguma sugestão de funcionalidade para o software, tanto em âmbito profissional como pessoal?

RESPOSTA

114 respostas em branco

- Queria saber se o produto se adapta facilmente a um setor industrial específico. A saúde pública carece bastante de gestão (amplamente falando) então se for flexível, talvez ajude até com a minha pesquisa de mestrado, se puder me enviar mais informações, agradeço. Mais pra frente eu pretendo buscar ferramentas de gestão de prontuários, estoques e equipamentos e que estes sejam visualizados como projetos dentro da cadeia de saúde, ou seja, englobando inúmeros stakeholders... caso se interesse podemos conversar melhor ☝ Parabénssss
- Módulo de aprimoramento de objetivos, abrindo espaço para comentários sobre defeitos e melhorias pelos demais usuários, algo na linha que evidencie o que precisa ser melhorado na execução de tarefas. Se entendi bem a funcionalidade do sistema, acredito que isso possa ser uma funcionalidade boa a mais.
- Controle de tarefas, decisão de tarefas por setores, temas e importância, gerenciamento de necessidades para finalizar a tarefa, cronograma de tarefas, anexação de fotos para evolução da tarefa, suporte para projetos e documentos essenciais
- O grande vácuo entre os softwares de gestão está na automação de processos, todos eles o usuário tem que tomar a ação ou repassar as atividades. Software que trabalha muito bem com isso é o Podio, que não esta na lista.
- não é bem uma funcionalidade mas acho legal ter indicações visuais quando termina uma tarefa, tipo “confetes” celebrando, por exemplo. me motiva mais a finalizar as coisas kkk
- Compartilhamento de agendas com os membros da equipe. Um local de canal de conversas transparente para todos os membros verem o que as áreas da empresa estão trabalhando
- Na agenda, ter dois tipos de compromissos, os obrigatórios e os não (participação optativa, caso haja disponibilidade)

- Por ser financeiro gostamos de ver o faturamento de cada projeto. Ae Otávio, chama no whats qqr coisa, Nery aqui.
- Opção para fazer chamada de vídeo (exemplo: zoom) podendo visualizar as informações inseridas na plataforma.
- Maior interatividade entre os membros de uma equipe, evitando os "feudos" que por exemplo o Trello cria.
- Implementar um processo estruturado de UX e UI para que se torne bem atrativo.
- Algo que misture a simplicidade do Trello mas com alguns recursos do Jira.
- Módulo para backlogs, controle de sprints e boards é requisito essencial
- Interface limpa e com gráficos das informações mais relevantes
- Descrição de demandas com prazo de entrega e responsável
- Tem que ter sincronização com o calendário do google!
- Relatório automático acerca das horas trabalhadas
- Fixação de prazos é o essencial, bem como dados
- Ele deve ter um workflow visível e organizado
- Procuro software mais simples e funcional.
- Notificações de prazos a serem cumpridos
- Quanto mais customizável melhor! #fdr
- App para mobile ajudaria bastante
- Controle de entregas da equipe
- Agenda. Controle de estoque.
- Chamadas com voz/video
- Filtro para fake news
- Nao pensei em algo

ANEXO B – Artigo

S.C.O.U.T - SISTEMA DE CONTROLE ORÇAMENTÁRIO E UTILIZAÇÃO DE TEMPO

Bernardo S. de Souza, Otávio C. Ribeiro, Martin G. Vigil

Departamento INE – Universidade Federal de Santa Catarina
Florianópolis, Brasil.

bschveitzer1101@gmail.com, otavio.carneiroribeiro@gmail.com, martin.vigil@ufsc.br

Abstract. *The electronic worksheet is a device used by companies to control the entry and exit time of their employees' working hours. Beside recording these hours, it helps to manage them, improving the work routines. The projects manager is also an instrument that makes the hours worked even more efficient, but not in the same way as the electronic worksheet. This way the idea was to unite the two tools so that a workday can be further optimized. And with the lack of a product that accomplishes this in the market, it was decided to develop software that reached this goal. The present final paper is a workday control system with an integrated project manager tool in a unique product, first called "SCOUT". The software was designed, developed and tested. To achieve the objectives proposed in this project, the methodology known as Design Science Research Methodology (DSRM) was used. The system was developed using programming languages for Web applications. And the tests prove that the system is functional, viable and has the potential to be a product.*

Resumo. *O ponto eletrônico é um dispositivo utilizado por empresas para controlar a hora da entrada e saída da jornada de trabalho de seus colaboradores. Além de registrar estas horas, ele auxilia na administração das mesmas, aperfeiçoando as rotinas de trabalho. O gerenciador de projetos também é um instrumento que torna as horas trabalhadas ainda mais eficientes, porém, não da mesma forma que o ponto eletrônico. Com isso, se teve a ideia de unir as duas ferramentas para que se possa otimizar ainda mais uma jornada de trabalho. E com a falta de um produto que realiza isso no mercado, foi decidido desenvolver um software que cumpre este objetivo. Neste trabalho de conclusão de curso é proposto um sistema de controle de jornada de trabalho com um sistema gerenciador de projetos integrado em um único produto, denominado inicialmente como "SCOUT", que significa Sistema de Controle Orçamentário e Utilização de Tempo. O sistema foi projetado, desenvolvido e testado. Para atingir os objetivos propostos neste trabalho, foi utilizada a metodologia conhecida por Design Science Research Methodology (DSRM). O sistema foi desenvolvido utilizando linguagens de programação para*

aplicações Web. E os testes provam que o sistema é funcional, é viável e tem potencial para ser um produto.

1. Introdução

Com a necessidade das empresas começarem a armazenar os horários de entrada, saída e até mesmo as pausas para o almoço de seus colaboradores, foi criado o relógio ponto, dispositivo mecânico que armazena estes horários em pedaços de papel pertencentes a cada colaborador. Com o passar do tempo, Barros (2019) explica que estes dispositivos começaram a se tornar sistemas informatizados e com isso o seu nome popular mudou para ponto eletrônico. Hoje eles são usados com registros marcados por biometria e até mesmo reconhecimento facial. Além disso, é possível armazenar os seus dados de maneira centralizada, podendo tirar conclusões de como realizar uma jornada de trabalho mais eficiente e ter um melhor controle destas horas.

Outra ferramenta que busca ajudar empresas a tornar as suas horas trabalhadas ainda mais eficientes é o gerenciador de projetos. Também na maioria das vezes eletrônico, o gerenciador de projetos auxilia no planejamento de cada projeto a fim de cumprir seus requisitos, alocando recursos e a programando suas atividades. Alguns exemplos deles são: Jira (ATLASSIAN, 2021), Trello (ATLASSIAN, 2021), Microsoft Project (MICROSOFT, 2021) e Podio (CITRIX SYSTEMS, 2021).

Apesar da grande ajuda que os gerenciadores de projetos trouxeram às empresas e até mesmo para pessoas que só querem administrar melhor as suas tarefas, um dos seus grandes problemas é a falta da funcionalidade de poder monitorar a quantidade de horas gastas para realização de cada atividade. Esse problema pode acarretar em erros para estipular o prazo da conclusão do projeto e também pode gerar gastos altos e desnecessários. Com isso, se foi pensado em planejar e desenvolver um sistema integrado de controle de jornada de trabalho (ponto eletrônico) e gerência de projetos de um jeito sucinto, de fácil utilização e que não exija uma grande curva de aprendizado dos usuários. Denominado inicialmente como "SCOUT", que significa Sistema de Controle Orçamentário e Utilização de Tempo. Neste sistema os colaboradores e seus gerentes teriam um acompanhamento em "tempo real" do projeto, mostrando os custos e esforços que já foram gastos em cada atividade, tarefa ou módulo. Podendo também gerar relatórios para acompanhar o projeto por inteiro, através do número total de horas gasta no projeto, gasta por colaborador e por determinada atividade. Com a união destes dois tipos de sistemas o usuário teria um maior controle e organização de suas horas trabalhadas, deixando-o mais seguro em relação aos problemas citados anteriormente.

2. Trabalhos Relacionados

Neste capitulo o objetivo foi identificar e descrever sobre alguns sistemas de gerenciamento de projetos e de ponto eletrônico encontrados no mercado.

O Trello é uma ferramenta de gerência de projetos que pode ser usada tanto para o acompanhamento de tarefas pessoais quanto para organizar projetos que envolvem equipes numerosas em grandes empresas. Nela o usuário pode criar quadros no qual são inseridas listas e dentro destas listas podem ser criados cartões com descrição, upload de arquivos multimídia, objetivos, entre outras coisas. Estes quadros podem ser compartilhados com qualquer pessoa que possui uma conta no Trello, com isso é possível atribuir usuários a cartões determinando que o mesmo é responsável por esta tarefa.

O sistema Kairós (DIMEP, 2021) segue a linha de um ponto eletrônico, onde o usuário registra os horários de entrada e saída, tanto para o início e final de expediente, quanto para os momentos de intervalo. Além do registro de horas, é possível determinar o tipo de cada jornada, ou seja, se foi realizada no ambiente de trabalho, alocado em algum cliente, alocado em cowork, alocado em homeoffice, entre outras opções. Os horários registrados tem que passar por uma aprovação de um usuário com permissões superiores, pode ser um gerente ou até mesmo supervisor da área. Após a aprovação, os horários registrados ficam à disposição de determinados usuários para consulta e geração de relatórios.

A plataforma Jira ajuda as equipes a planejar, atribuir, acompanhar, criar relatórios e gerenciar o trabalho, unindo as equipes em tudo, desde o desenvolvimento de software e suporte ao cliente com agilidade. O Jira pode ser usado com a metodologia Scrum, Kanban ou os dois juntos de uma forma mista. Fornece estimativas que ajudam a equipe a se tornar mais precisa e eficiente de uma forma transparente para que toda a equipe se mantenha a par de tudo.

A Asana (ASANA, 2021) é uma plataforma de gestão de projetos e equipes onde é possível criar projetos com um conjunto de tarefas a serem executadas que podem ou não seguir uma ordem pré-definida. As tarefas e fluxos de atividades que podem ser compartilhados com membros do time e nelas podem ser definidos prazos e responsáveis. Segue a metodologia ágil kanban (RADIGAN, 2021), podendo passar a atividades de uma coluna para outra conforme vão sendo finalizadas. Além da visualização por colunas, se pode usar a forma de lista de tarefas, cronograma ou calendário.

3. Metodologia

Para atingir os objetivos deste projeto será utilizada a metodologia conhecida por Design Science Research Methodology (DSRM) que é um método de projetar artefatos para resolver problemas, e a mesma contém as seguintes etapas. A primeira etapa trata da identificação do problema e sua causa e já foi executada e documentada na introdução deste projeto. A segunda etapa é onde definem-se os objetivos da solução. Em seguida, é projetado e desenvolvido um artefato para solucionar o problema proposto. Depois, na quarta etapa, o artefato é demonstrado para solucionar o problema e por fim, na quinta etapa, será avaliado para garantir que atinja os objetivos da segunda etapa (JUNIOR et al., 2017). A seguir, serão descritas cada uma das etapas citadas acima, excluindo a primeira por ela já estar descrita, e por fim um resumo da metodologia e quais etapas cumprem cada objetivo específico do projeto.

Definir objetivos da solução tem como meta o desenvolvimento de um software que una as funcionalidades de uma ferramenta de ponto eletrônico com um gerenciador de projetos. A definição dessa solução foi iniciada buscando referências do estado da arte de aplicações de ponto eletrônico e de gerência de projetos, focando nas que possibilitam um uso fácil, com interface amigável e configuração intuitiva. Após isso será estudado e decidido quais tecnologias serão utilizadas para a produção do artefato. E para finalizar, será executada uma pesquisa de mercado para coletar opiniões relacionadas ao problema levantado e a solução proposta.

Planejar e desenvolver o artefato, será criado um artefato de software para solucionar o problema. Primeiro será feita a análise de requisitos do projeto, com ela, será identificada todas as funcionalidades que o artefato deverá ter. Depois será levando os tipos de usuário

que existirão no software e as suas permissões. Em seguida será feito o protótipo das telas que ajudarão a entender e desenvolver os fluxos do sistema. Após finalizar essas etapas, será iniciada a implementação do artefato baseado nos requisitos e protótipos concebidos anteriormente. A implementação será dividida em duas fases, o desenvolvimento do servidor e desenvolvimento da interface web para usuários.

Demonstrar o uso do artefato para solucionar o problema, onde Serão realizados testes para validar se o artefato resolve o problema conforme o planejado e se cumpre os seus requisitos. Os testes serão do tipo funcionais e serão executados manualmente. Primeiramente, será testada cada parte do artefato individualmente e depois todas as partes juntas como um todo. Com isso, será capaz de medir a qualidade e a confiabilidade do artefato.

4. Resultados

Neste capítulo serão apresentados os resultados. O SCOUT foi desenvolvido e abaixo é apresentado as suas principais telas e funcionalidades.

4.1. Principais Telas do Sistema

Nesta sessão serão apresentadas e detalhadas as principais telas do sistema. A tela de Autenticação é composta pelos campos de usuário e senha, como mostrado na Figura 1.

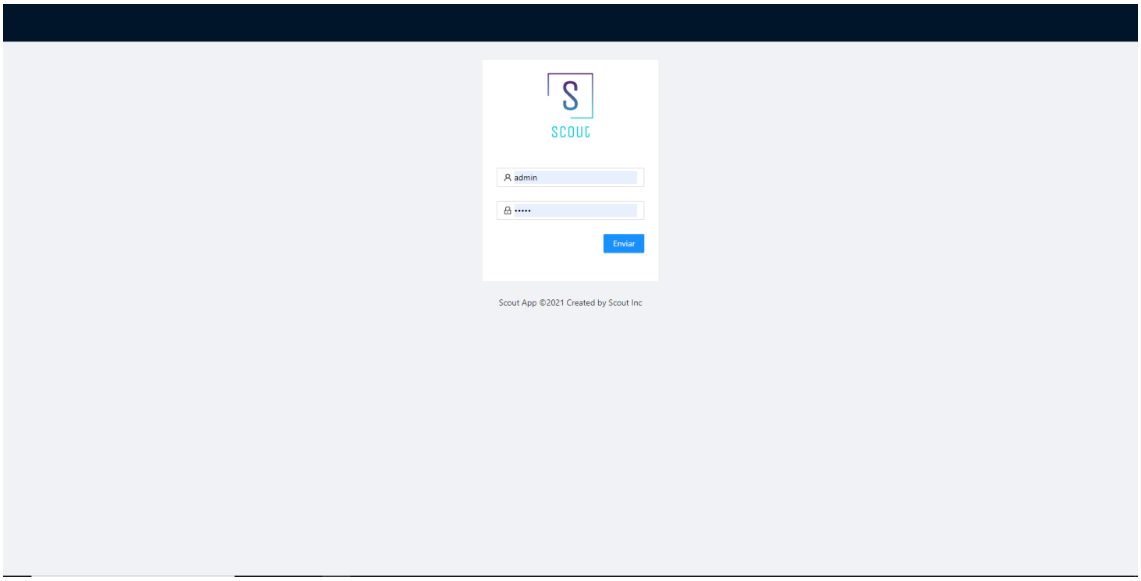


Figura 1 – Tela de Autenticação

A tela Inicial/Home é composta pela data, botão para iniciar a função de Ponto Eletrônico, botão para pausar a função de Ponto Eletrônico. Os últimos lançamentos do usuário, que são os horários que o usuário trabalhou em um determinado dia e as tasks que o mesmo executou. E por fim, uma lista das tasks atribuídas ao usuário. Esta estrutura é mostrada na Figura 2. Também na tela Inicial/Home é encontrado um menu lateral que leva o usuário para as demais telas, como mostrado de modo expandido na Figura 3.

Figura 2 – Tela Inicial/Home

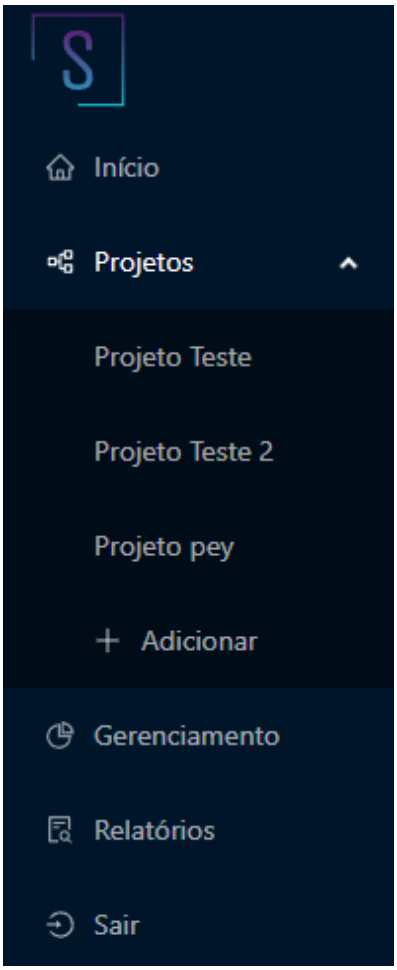


Figura 3 – Menu lateral expandido

A tela de Projeto, mostrada na Figura 4, é composta pelas informações do projeto, contendo data de criação, usuários do projeto, identificando quem é o administrador e uma lista de tasks vinculadas ao projeto. Ela é composta também do botão “Editar Projeto”, no qual abre-se um modal em que o usuário pode alterar as informações do projeto como mostrado na Figura 5. O botão de “Gerenciar membros do projeto”, habilitado apenas para o administrador, no qual abre-se um modal mostrado na Figura 6 e o botão de “Adicionar Task”, no qual abre-se um modal mostrado na Figura 7.

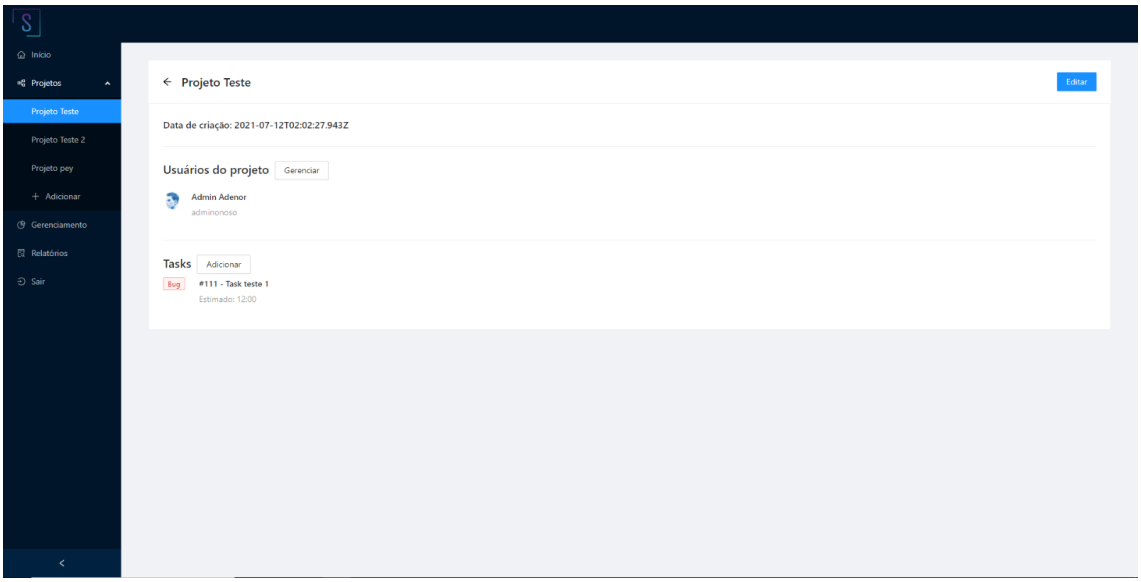


Figura 4 – Tela de Projeto

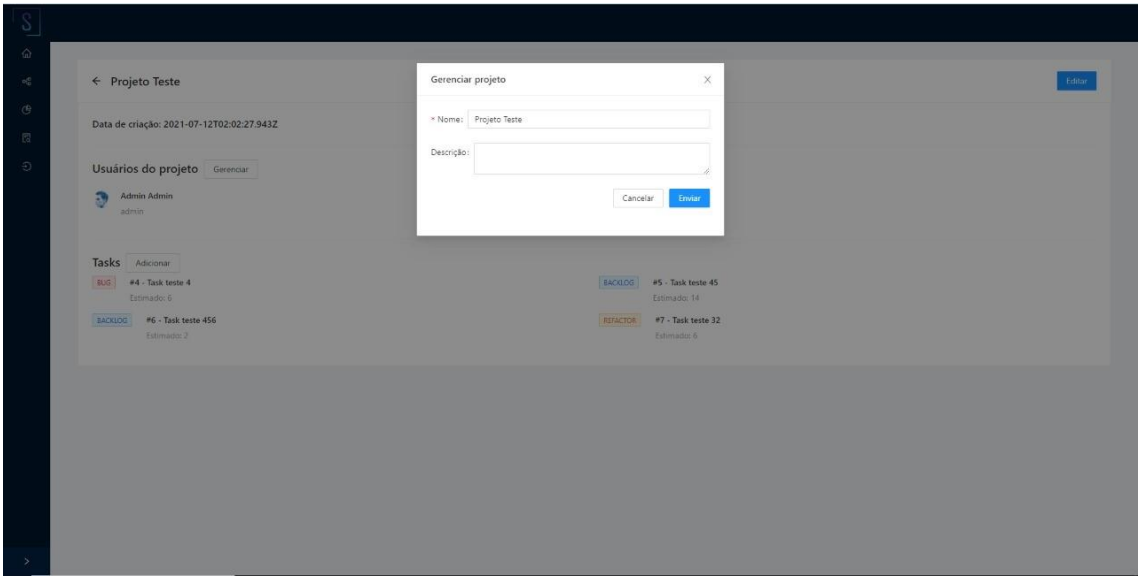


Figura 5 – Modal Editar Projeto

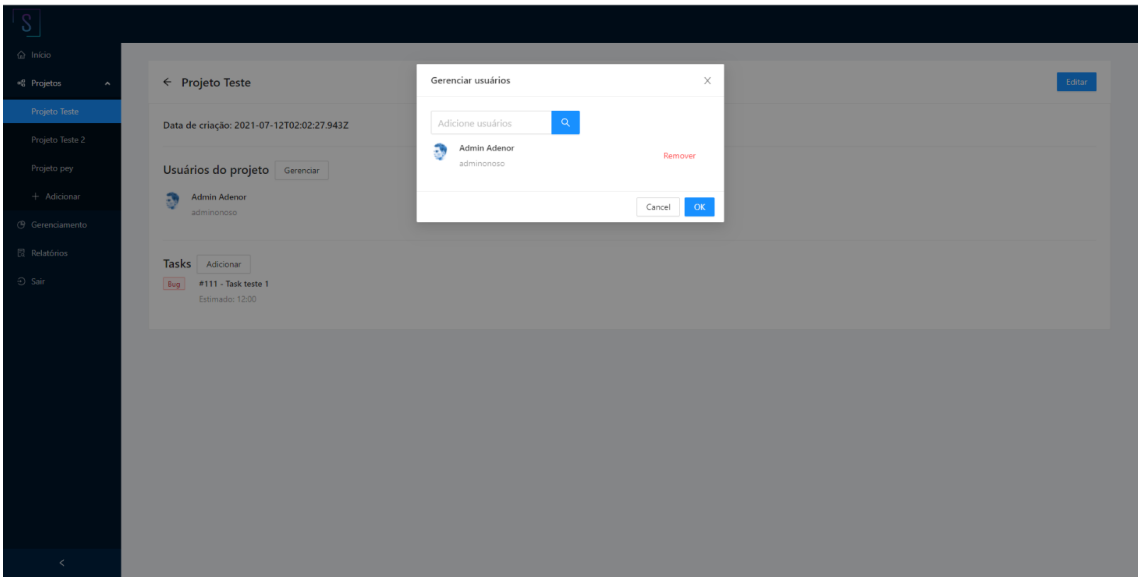


Figura 6 – Modal de Gerenciar Membros do Projeto

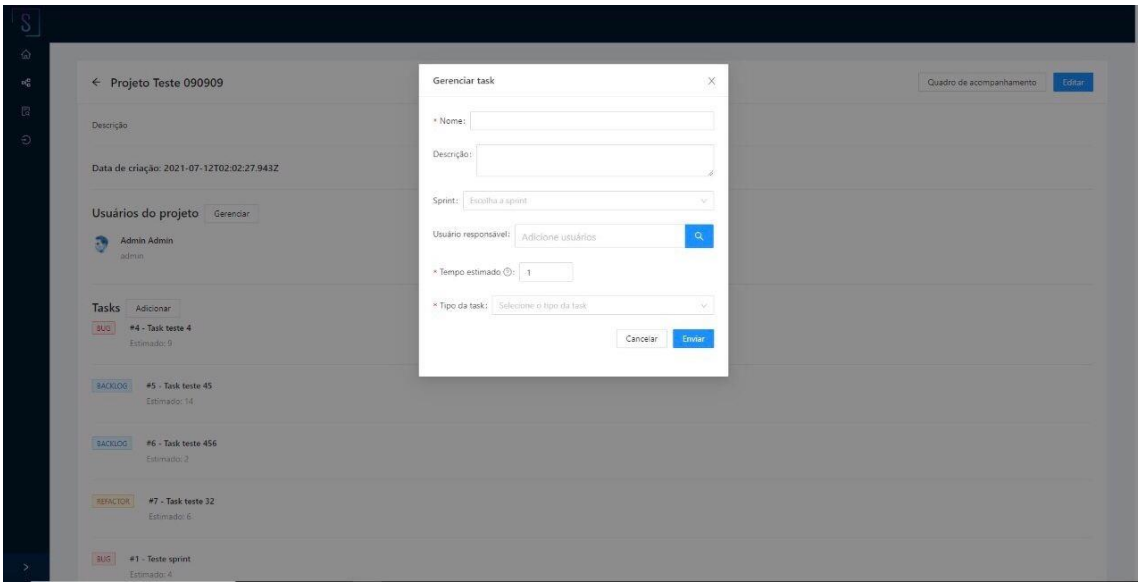


Figura 7 – Modal Criação de Task

A tela de Quadro de Acompanhamento é composta de um quadro visual de um determinado projeto com as colunas “Em Aberto”, “Executando”, “Testando” e “Finalizado”. Onde as tasks são organizadas automaticamente dependendo do estado de cada uma como mostrado na Figura 8.

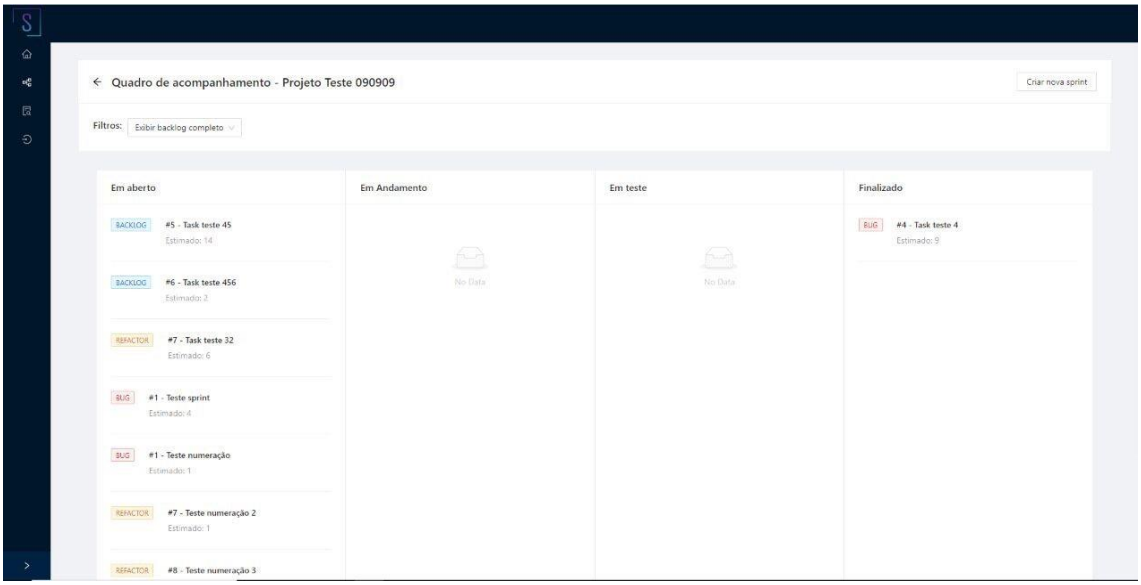


Figura 8 – Tela de Quadro de Acompanhamento

A tela de Relatório de Horas, mostrada na Figura 9, permite visualizar as horas trabalhadas pelo usuário no mês que o usuário filtra. A tela é composta de um calendário onde cada dia é possível selecionar as horas trabalhadas. Ao selecionar estas horas, é aberto um modal onde se pode visualizar o intervalo de horas trabalhadas naquele dia e as tasks que o usuário estava trabalhando naquele intervalo de horas, como mostrado na Figura 10. E também, ao selecionar um dia do calendário, é aberto o modal de "Informações Diárias". Esse cadastro é feito manualmente. Ao selecionar o botão "Adicionar novo intervalo", o sistema preenche o campo de hora inicial automaticamente com o horário real, sendo que o usuário pode alterá-lo. O horário final fica em branco, mas o usuário tem a opção de preenchê-lo, caso ele já tenha feito estas horas anteriormente. O modal de "Informações Diárias" é mostrado na Figura 11.

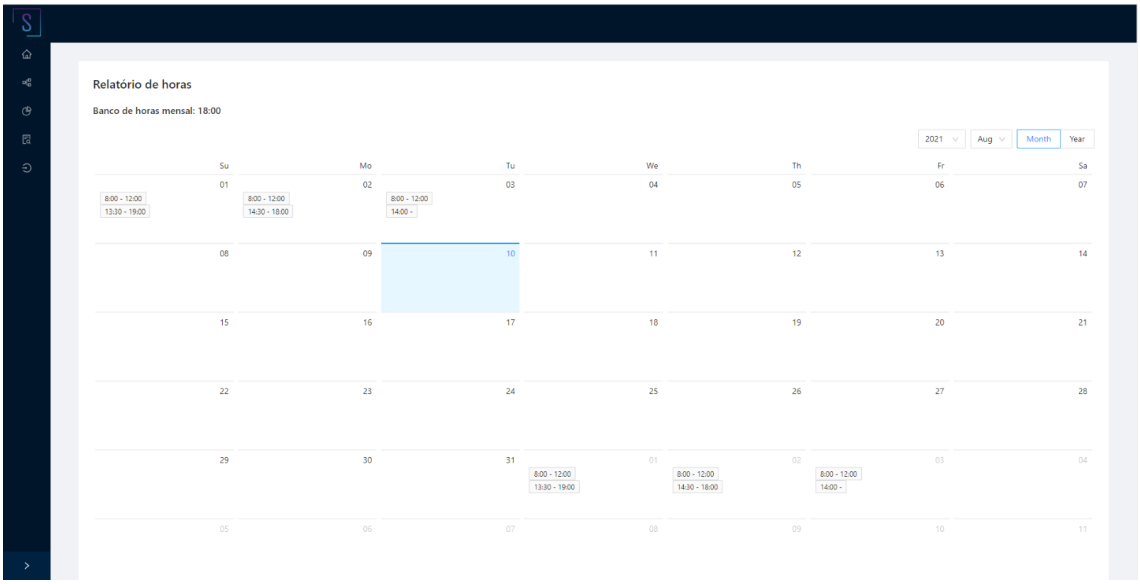


Figura 9 – Tela de Relatório de Horas

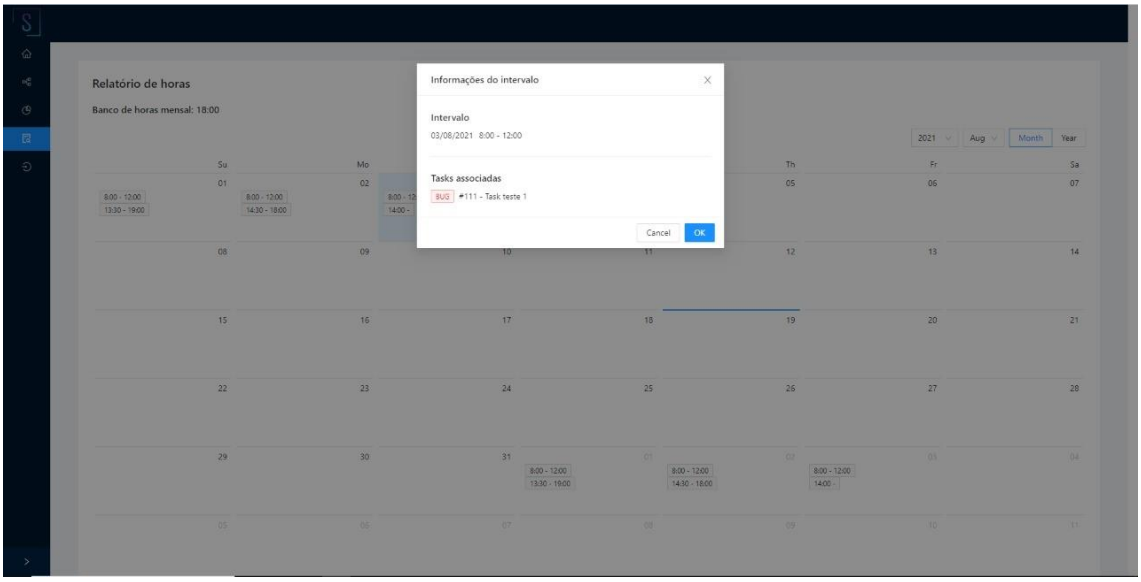


Figura 10 – Modal Visualizar Horas

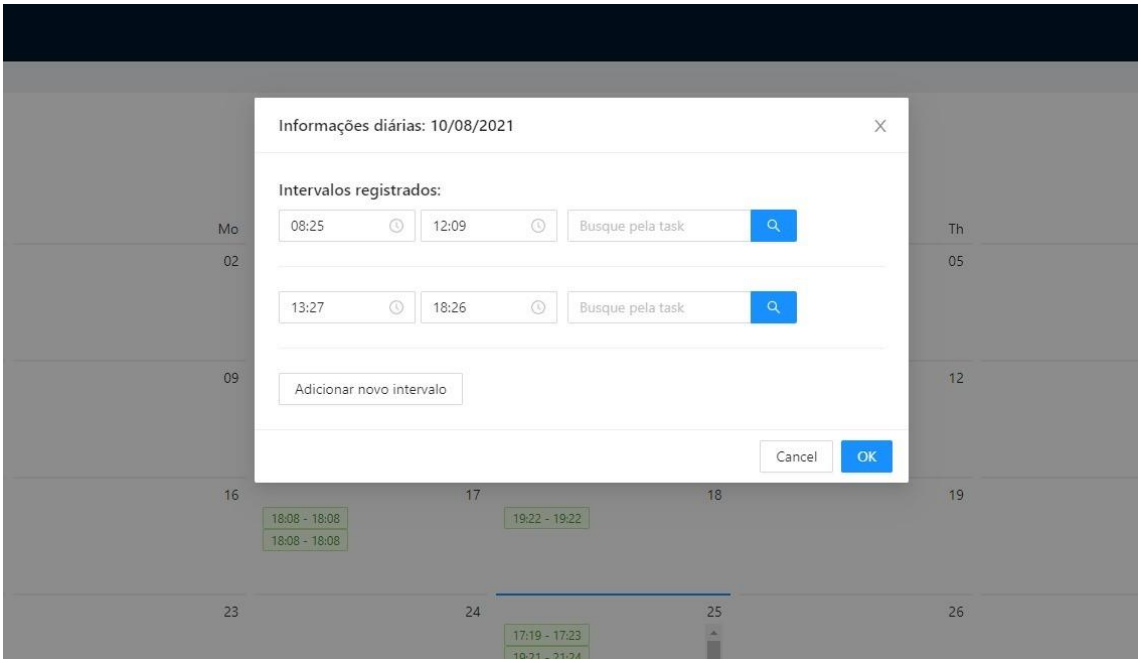


Figura 11 – Modal Informações Diárias

5. Avaliação

Para avaliar o SCOUT e verificar que o mesmo cumpre todos os requisitos impostos a ele com o menor número de erros possíveis, foram realizados testes funcionais. Os mesmos são executados analisando as saídas do sistema, perante entradas pré-determinas. Também são conhecidos como teste de caixa-preta por não terem contato com o código,

mas somente com o programa ou parte dele (WAZLAWICK, 2013). Foi testada cada parte do sistema individualmente e depois todas as partes juntas como um todo. Com isso, foi capaz de medir a qualidade e a confiabilidade da aplicação.

Para auxiliar na organização da execução destes testes, foram criados uma série de casos de teste. Eles servem para documentar as entradas, o que acontece e as saídas esperadas de uma determinada funcionalidade do sistema. E o seu principal objetivo é encontrar erros. Abaixo, está sendo mostrado alguns dos principais casos de teste:

ID: 4

Caso de Teste: Validar se hora está sendo registrada com sucesso.

Tela: Relatório de Horas.

Pré-requisito: Estar logado no sistema com qualquer tipo de usuário.

Passos de teste:

1 – Logar no sistema com qualquer tipo de usuário.

2 – Acessar tela de "Relatório de Horas".

3 – Selecionar dia que o usuário quer registrar a hora.

4 – No modal de Informações diárias, selecionar o botão "Adicionar novo intervalo".

5 – Preencher as horas e selecionar o botão "Ok"

Resultados esperados: Sistema emite toast com a mensagem "Horas registradas com sucesso" e a hora é mostrada no dia selecionado pelo usuário no Passo 3.

Resultado atual: Sistema emite toast com a mensagem "Horas registradas com sucesso" e a hora é mostrada no dia selecionado pelo usuário no Passo 3.

Status: OK

ID: 7

Caso de Teste: Validar se projeto está sendo adicionado com sucesso.

Tela: Adicionar Projeto.

Pré-requisito: Estar logado no sistema com um usuário com perfil Administrador.

Passos de teste:

1 – Logar no sistema com um usuário com perfil Administrador.

2 – Acessar tela de "Adicionar projeto".

3 – Preencher campos obrigatórios.

4 – Selecionar o botão "Adicionar".

Resultados esperados: Sistema emite toast com a mensagem "Projeto adicionado com sucesso" e é mostrado a tela do projeto criado.

Resultado atual: Sistema emite toast com a mensagem "Projeto adicionado com sucesso" e é mostrado a tela do projeto criado.

Status: OK

ID: 19

Caso de Teste: Validar se status da task muda ao gerenciar task.

Tela: Modal Gerenciar Task.

Pré-Requisitos: Estar logado no sistema com um usuário que seja membro do projeto que a task faz parte.

Passos de teste:

1 - Logar no sistema com um usuário que seja membro do projeto que a task faz parte.

2 – Selecionar projeto no qual a task faz parte.

3 – Selecionar a task que se deseja mudar o status.

4 – Selecionar o botão “Gerenciar task”.

5 – Alterar Status da task.

6 – Selecionar o botão “Enviar”.

Resultado esperado: Sistema emite toast com a mensagem "Task atualizada com sucesso" e status da task é alterado.

Resultado atual: Sistema emite toast com a mensagem "Task atualizada com sucesso" e status da task não é alterado.

Status: Não OK.

Alguns erros foram encontrados, como o do caso de teste id 19 acima. Os mesmos foram registrados e corrigidos. Depois os testes eram refeitos até que todos os casos de teste estivessem com o status OK. Segue o registro de erro do caso de teste id 19:

Nome: Aplicação não muda status da task ao gerencia-la no modal de "Gerenciar Task".

Descrição: Aplicação não muda o status da task no banco de dados após alterar o campo Status no modal de “Gerenciar Task”.

Passos para reproduzir:

1 - Logar no sistema com um usuário que seja membro do projeto que a task faz parte.

2 – Selecionar projeto no qual a task faz parte.

3 – Selecionar a task que se deseja mudar o status.

4 – Selecionar o botão “Gerenciar task”.

5 – Alterar Status da task.

6 – Selecionar o botão “Enviar”.

Evidencias: Após a emissão do toast com a mensagem "Task atualizada com sucesso", o status task não é atualizada no banco de dados.

Critérios de aceitação: Após a emissão do toast com a mensagem "Task atualizada com sucesso", o status task precisa ser atualizado no banco de dados e na interface.

Após finalizar os testes funcionais, o desenvolvimento do SCOUT foi concluído.

6. Discussão

O planejamento e o desenvolvimento do SCOUT em um modo geral foram trabalhosos, porém tranquilos de serem executados. Se teve tempo suficiente para fazer a organizar e cumprir o cronograma proposto no início do planejamento. A equipe trabalhou muito bem junta, sempre ajudando um ao outro e trazendo discussões interessantes para as reuniões, onde se era debatido várias formas de execução do projeto. As tecnologias escolhidas para o desenvolvimento do SCOUT ajudaram bastante, por serem novas tecnologias e de fácil manipulação. Durante os processos não houveram grandes limitações, porém uma delas foi o fato de não botar a aplicação rodando na nuvem que não permitiu a execução da avaliação com usuários reais. Apesar dessa limitação, o planejamento e desenvolvimento do SCOUT foram feitos de forma muito satisfatória, trazendo muitos ensinamentos e aprendizados para os autores.

7. Considerações finais

Neste trabalho de conclusão de curso, foi proposto um software que une uma ferramenta de gerenciamento de projeto com as funcionalidades de um ponto eletrônico que atualiza automaticamente as informações nos projetos dentro do sistema e gera relatórios de horas, para melhor controle da jornada de trabalho. O sistema foi nomeado de SCOUT", que significa Sistema de Controle Orçamentário e Utilização de Tempo. Após a proposta ter sido aprovada, o este trabalho começou a ser planejado com a ajuda da metodologia conhecida por Design Science Research Methodology (DSRM). Após isso, foi realizada uma pesquisa de mercado com o objetivo de saber se este software era de interesse das pessoas. Se obteve uma resposta positiva e ainda se pôde localizar quais produtos presentes no mercado atual que mais se assemelhavam com o SCOUT. Em seguida, depois de se estudar sobre estes produtos, o desenvolvimento do sistema foi iniciado e concluído. As principais tecnologias utilizadas foram React com a linguagem de programação Javascript para o front-end e, Node e Express também com a linguagem Javascript para o back-end. Por fim, foram realizados testes funcionais no software e foi concluído que ele é funcional, viável e tem potencial para se tornar um produto.

8. Referências bibliográficas

Barros, L. O que fazer quando sua máquina de bater ponto é ultrapassada? 2019. Disponível em: https://blog.tangerino.com.br/maquina-de-bater-ponto-ultrapassada/> Acesso em: 25 jun, 2021

ATLASSIAN. Jira Software - Features. 2021. Disponível em: https://www.atlassian.com/software/jira/features/> Acesso em: 2 jun, 2021.

ATLASSIAN. Trello Tour. 2021. Disponível em: https://trello.com/tour/> Acesso em: 2 jun, 2021.

MICROSOFT. Microsoft Project. 2021. Disponível em: https://www.microsoft.com/pt-br/microsoft-365/project/project-management-software/> Acesso em: 2 jun, 2021.

CITRIX SYSTEMS. Tour - Citrix Podio. 2021. Disponível em: https://podio.com/site/tour/> Acesso em: 2 jun, 2021.

DIMEP. Kairos. 2021. Disponível em: https://www.dimep.com.br/kairos/> Acesso em: 22 jun, 2021

ASANA. Software de gerenciamento. 2021. Disponível em: https://asana.com/pt/uses/project-management/> Acesso em: 22 jun, 2021

Radigan, D. Kanban - A brief introduction. 2021. Disponível em: https://www.atlassian.com/agile/kanban/> Acesso em: 25 jun, 2021

JUNIOR, V. F. et al. Design science research methodology enquanto estratégia metodológica para a pesquisa tecnológica. Revista Espacios, v. 38, n. 6, p. 25–34, 2017.

Wazlawick, R. S. Engenharia de Software: Conceitos e práticas. Florianópolis: Elsevier, 2013.

ANEXO C – Código

Backend

AuthController.js

```
const jwt = require('jsonwebtoken')

module.exports = {
  verifyAuth: async (req, res, next) => {
    const bearerHeader = req.headers['authorization']

    if(!bearerHeader) return res.sendStatus(403)

    const bearer = bearerHeader.split(' ')

    const bearerToken = bearer[1]

    jwt.verify(bearerToken, 'scoutKey', (err, authData) => {
      if(err) return res.sendStatus(403)

      if(req.path === '/user/me') return res.json(authData)

      next()
    })

  }
}
```

OpenController.Js

```
const mongoose = require('mongoose')
const User = mongoose.model('User')
const jwt = require('jsonwebtoken')
const error = require('../utils/errors')
const emailService = require('../utils/emailService')
const moment = require('moment')


module.exports = {
  login: async (req, res) => {
    try {
```

```
      const userInfo = await User.findOne(req.body, '-password
-createdAt -updatedAt').populate('tasks')

      if(!userInfo) return
res.status(400).json(error.loginErrors['NotFound'])

      jwt.sign({ userInfo }, 'scoutKey', (err, token) => {
        res.json({ userInfo, token })
      })

    } catch (err) {
      return res.status(400).json(err)
    }
  },

  createUser: async (req, res) => {
    try {
      req.body.password = Math.floor(1000 + Math.random() * 9000)
      const user = await User.create(req.body)
      return res.json(user)
    } catch (err) {
      const responseErr = error.crudUserErrors[err.name] ?
error.crudUserErrors[err.name] : err
      return res.status(400).json(responseErr)
    }
  },

  recoverPassword: async (req, res) => {
    try {
      const user = await User.findOne(req.body, 'login')
      const timestamp = moment().unix()

      if(!user) return
res.status(400).json(error.recoverPasswordErrors['NotFound'])

      const link =
`${process.env.FRONT_URL}/resetPassword/${user._id}%${timestamp}`

      let info = await emailService.transporter.sendMail({
        from: '"Não responda" <foo@example.com>', // Endereço do
rementente
        to: user.login, // Lista de recebedores
        subject: "Recuperação de senha", // Linha do assunto
```

```
        text: "Hello world?", // Conteudo sem HTML
        html: `<a href=${link}>Clique aqui</a>` // Conteudo com HTML
      });

      if(info.accepted.length === 0) return
res.status(400).json(error.recoverPasswordErrors['SendingError'])

      await User.findByIdAndUpdate(user._id, { recoverPassword:
timestamp })

      return res.json({ message: 'Solicitação de recuperação de senha
enviada. Verifique seu e-mail.'})
    } catch (err) {
      return res.status(400).json(err)
    }
  },

  resetPassword: async (req, res) => {
    try {
      const { timestamp, password } = req.body
      const user = await User.findById(req.params.id,
'recoverPassword')
      if(!user) return
res.status(400).json(error.recoverPasswordErrors['InvalidId'])

      const timestampDiff =
moment(timestamp).diff(user.recoverPassword, 'hours')
      if(timestampDiff > 23 || !user.recoverPassword) return
res.status(400).json(error.recoverPasswordErrors['ExpiredSolicitation']
)

      await User.findByIdAndUpdate(req.params.id, { recoverPassword:
'', password })

      return res.json({ message: 'Senha atualizada com sucesso.'})
    } catch (err) {
      return res.status(400).json(err)
    }
  }
}
```

ProjectController.js

```
const mongoose = require('mongoose')
```

```
const Project = mongoose.model('Project')
const User = mongoose.model('User')
const Sprint = mongoose.model('Sprint')
const error = require('../utils/errors')

module.exports = {
  getProjectDetails: async (req, res) => {
    const project = await
Project.findById(req.params.projectId).populate('users').populate('task
s').populate('sprints')
    return res.json(project)
  },
  getProjectsFromOrg: async (req, res) => {
    const { page = 1 } = req.query
    const projects = await Project.paginate({ organization:
req.params.organizationId }, { page, limit: 10 })
    return res.json(projects)
  },
  createProject: async (req, res) => {
    try {
      const project = await Project.create(req.body)
      return res.json(project)
    } catch (err) {
      return res.status(400).json(err)
    }
  },
  updateProject: async (req, res) => {
    try {
      const project = await
Project.findByIdAndUpdate(req.params.projectId, req.body, { new: true
}).populate('users').populate('tasks').populate('sprints')
      return res.json(project)
    } catch (err) {
      return res.status(400).json(err)
    }
  },
  addUserProject: async (req, res) => {
    try {
      const project = await
Project.findByIdAndUpdate(req.params.projectId, { $addToSet: { 'users':
req.body.userId } }, { new: true
}).populate('users').populate('tasks').populate('sprints')
      return res.json(project)
```

```
    } catch (err) {
      return res.status(400).json(err)
    }
  },
  removeUserProject: async (req, res) => {
    try {
      const project = await
Project.findByIdAndUpdate(req.params.projectId, { $pull: { 'users':
req.body.userId } }, { new: true
}).populate('users').populate('tasks').populate('sprints')
      return res.json(project)
    } catch (err) {
      return res.status(400).json(err)
    }
  },
  createNewSprint: async (req, res) => {
    try {
      const maxSprint = await Sprint.find({project:
req.params.projectId}).sort({number:-1}).limit(1);
      const number = maxSprint.length ? maxSprint[0].number + 1 : 1;
      const newSprintObj = {
        name: 'Sprint ' + number,
        number,
        project: req.params.projectId
      }

      const newSprint = await Sprint.create(newSprintObj);

      const projectUpdated = await
Project.findByIdAndUpdate(req.params.projectId, { $addToSet: {
'sprints': newSprint._id } }, { new: true })
      .populate('users')
      .populate('sprints')
      .populate('tasks')

      return res.json(projectUpdated)
    } catch (err) {
      return res.status(400).json(err)
    }
  },
  filterTasksBySprint: async (req, res) => {
    try {
```

```
      const filteredTasks = await Sprint.findOne({_id:
req.params.sprintId, project: req.params.projectId}).populate('tasks')
      return res.json(filteredTasks)
    } catch(err) {
      return res.status(400).json(err)
    }
  }
}
```

ReportController.js

```
const mongoose = require('mongoose')
const UserMonthReport = mongoose.model('UserMonthReport')
const UserDayReport = mongoose.model('UserDayReport')
const Interval = mongoose.model('Interval')
const Task = mongoose.model('Task')
const error = require('../utils/errors')
const moment = require('moment')

module.exports = {
  updateUserHoursReport: async (req, res) => {
    const { user, days } = req.body

    try {
      await days.map((dayIndex) => {
        dayIndex.intervals.map(async (interval) => {
          const objInterval = { ...interval, user }
          delete objInterval.id

          if(interval.id) {
            const updateInterval = await Interval.findByIdAndUpdate(
              interval.id,
              objInterval
            )
          } else {
            const createInterval = await Interval.create(objInterval)
            const dayReport = await UserDayReport.update(
              { month: req.params.monthId, dayOrder: dayIndex.day},
              { $push: { intervals: createInterval._id}},
              { upsert: true }
            )

            if(dayReport.upserted && dayReport.upserted.length > 0) {
              await UserMonthReport.findByIdAndUpdate(
```

```
                req.params.monthId,
                { $push: { dailyReports: dayReport.upserted[0]._id } }
              )
            }

          }
        })
      })

      res.json({ message: 'Horários atualizados com sucesso!'})

    } catch(err) {
      res.status(400).json(err)
    }
  },

  verifyMonthReport: async (req, res) => {
    const { user } = req.params
    try {
      const monthReport = await UserMonthReport.findOne({
        monthOrder: moment().month(),
        year: moment().year(),
        user: user
      })

      if(!monthReport) {
        const createMonth = {
          name: moment().month(moment().month()).format('MMMM'),
          monthOrder: moment().month(),
          dailyReports: [],
          year: moment().year(),
          user
        }
        const newMonthReport = await
UserMonthReport.create(createMonth)
        res.json(newMonthReport)
      } else {
        res.json(monthReport)
      }
    } catch (err) {
      res.status(400).json(err)
    }
  },
```

```
  getMonthReport: async (req, res) => {
    try {
      const monthReport = await UserMonthReport.findOne({user:
req.params.userId, monthOrder: parseInt(req.params.month), year:
parseInt(req.params.year) })
        .populate({
          path: 'intervals',
          populate: {
            path: 'task'
          }
        })
      res.json(monthReport)
    } catch (err) {
      res.status(400).json(err)
    }

  },

  getUserReports: async (req, res) => {
    const { page = 1 } = req.query
    try {
      const userReports = await UserMonthReport.paginate(req.params, {
page, limit: 10 })
      res.json(userReports)
    } catch (err) {
      res.status(400).json(err)
    }
  },

  updateIntervals: async (req, res) => {
    try {
      const intervals = req.body.intervals;
      const monthOrderNum = moment(req.body.originalDate).month();
      const yearNum = moment(req.body.originalDate).year();

      for(let index = 0; index < intervals.length; index++) {
        let updateObj = {...intervals[index]};

        updateObj.dayOrder = req.body.dayOrder;

        if(intervals[index].task) {
          updateObj.task = intervals[index].task.value;
```

```
        }


        if(updateObj._id) {
          await Interval.findByIdAndUpdate(updateObj._id, updateObj, {
new: true });
        } else {
          const newInterval = await Interval.create(updateObj);

          const updatedReport = await UserMonthReport.findOneAndUpdate(
            {user: req.body.user._id, monthOrder: monthOrderNum, year:
yearNum},
            { $addToSet: { 'intervals': newInterval._id } }
          )

          const monthlyReportObj = {
            user: req.body.user._id,
            monthOrder: monthOrderNum,
            year: yearNum,
            intervals: [newInterval._id]
          }

          if(!updatedReport) {
            await UserMonthReport.create(monthlyReportObj);
          }
        }
      }

      const monthlyReport = await UserMonthReport.findOne({user:
req.body.user._id, monthOrder: monthOrderNum, year: yearNum})
        .populate({
          path: 'intervals',
          populate: {
            path: 'task'
          }
        })
      res.json(monthlyReport)
    } catch (err) {
      res.status(400).json(err)
    }
  },
}
```

SprintController.js

```
const mongoose = require('mongoose')
const Sprint = mongoose.model('Sprint')
const error = require('../utils/errors')

module.exports = {
  getSprintsFromProject: async (req, res) => {
    const { page = 1 } = req.query
    const sprints = await Sprint.paginate({ project: req.params.id }, {
page, limit: 10 })
    return res.json(sprints)
  },
  getActiveSprint: async (req, res) => {
    const { page = 1 } = req.query
    const sprints = await Sprint.paginate({ project: req.params.id,
active: true }, { page, limit: 10 })
    return res.json(sprints)
  },
  createSprint: async (req, res) => {
    try {
      const sprint = await Sprint.create(req.body)
      return res.json(sprint)
    } catch (err) {
      return res.status(400).json(err)
    }
  },
  updateSprint: async (req, res) => {
    try {
      const sprint = await Sprint.findByIdAndUpdate(req.params.id,
req.body, { new: true })
      return res.json(sprint)
    } catch (err) {
      return res.status(400).json(err)
    }
  }
}
```

TaskController.js

```
const mongoose = require('mongoose')
const Task = mongoose.model('Task')
const User = mongoose.model('User')
const Project = mongoose.model('Project')
const Interval = mongoose.model('Interval')
```

```
const Sprint = mongoose.model('Sprint')
const error = require('../utils/errors')

module.exports = {
  getTasksFromProject: async (req, res) => {
    try {
      const { page = 1 } = req.query
      const tasks = await Task.paginate({ project: req.params.id }, {
page, limit: 10 })
      return res.json(tasks)
    } catch (err) {
      return res.status(400).json(err)
    }
  },
  createTask: async (req, res) => {
    try {
      const preparedTask = await
module.exports.prepareTaskToCreate(req.body.task, req.body.project);
      const task = await Task.create(preparedTask)

      if(task.user) {
        await User.findByIdAndUpdate(task.user, { $addToSet: { 'tasks':
task._id } })
      }

      if(task.sprint) {
        await Sprint.findByIdAndUpdate(task.sprint, { $addToSet: {
'tasks': task._id } })
      }

      await Project.findByIdAndUpdate(req.body.project, { $addToSet: {
'tasks': task._id } })

      return res.json(task)
    } catch (err) {
      return res.status(400).json(err)
    }
  },
  prepareTaskToCreate: async (task, project) => {
    try {
      const projectObj = await Project.findById(project)
```

```
      task.number = projectObj.tasks.length ? projectObj.tasks.length +
1 : 1;
      task.status = 'open';

      return task;
    } catch (err) {
      return err
    }
  },
  updateTask: async (req, res) => {
    try {
      const task = await Task.findByIdAndUpdate(req.params.taskId,
req.body, { new: true })
      return res.json(task)
    } catch (err) {
      return res.status(400).json(err)
    }
  },
  removeTask: async (req, res) => {
    try {
      const task = await Task.findByIdAndRemove(req.params.id);
      if(task.parentTask) {
        await Task.findByIdAndUpdate(task.parentTask, { $pull: { tasks:
req.params.id } })
      }
      await Interval.findOneAndUpdate({ task: req.params.id }, { task:
'' })
      return res.json(task)
    } catch (err) {
      return res.status(400).json(err)
    }
  },
  getTaskById: async (req, res) => {
    try {
      const task = await
Task.findById(req.params.taskId).populate('user').populate('subtasks').
populate('parentTask').populate('sprint')
      const project = await Project.findOne({ tasks: req.params.taskId
}).populate('sprints')

      return res.json({task, project})
    } catch (err) {
      return res.status(400).json(err)
```

```
    }
  },
  addUserToTask: async (req, res) => {
    try {
      const task = await Task.findByIdAndUpdate(req.params.taskId, {
user: req.body.user }, { new: true }).populate('user')
      await User.findByIdAndUpdate(req.body.user, { $addToSet: { tasks:
req.params.taskId } })
      return res.json(task)
    } catch (err) {
      return res.status(400).json(err)
    }
  },
  removeUserFromTask: async (req, res) => {
    try {
      const task = await Task.findByIdAndUpdate(req.params.taskId, {
user: null })
      await User.findByIdAndUpdate(req.body.user, { $pull: { tasks:
req.params.taskId } })
      return res.json(task)
    } catch (err) {
      return res.status(400).json(err)
    }
  },
  setTaskToInterval: async (req, res) => {
    try {
      const interval = await
Interval.findByIdAndUpdate(req.body.intervalId, { $pull: { task:
req.params.id } })
      return res.json(interval)
    } catch (err) {
      return res.status(400).json(err)
    }
  },
  removeTaskFromInterval: async (req, res) => {
    try {
      const interval = await
Interval.findByIdAndUpdate(req.body.intervalId, { $pull: { task: '' }
})
      return res.json(interval)
    } catch (err) {
      return res.status(400).json(err)
    }
```

```
  },
  createSubtask: async (req, res) => {
    try {
      const subtask = await Task.create(req.body)
      await Task.findByIdAndUpdate(subtask.parentTask, { $push: {
subtasks: subtask.id } })
      return res.json(subtask)
    } catch (err) {
      return res.status(400).json(err)
    }
  },
  updateTaskSprint: async (req, res) => {
    try {
      await Sprint.findOneAndUpdate({tasks: req.params.taskId},  {
$pull: { 'tasks': req.params.taskId } })

      let newTask;

      if(req.body.sprint === 'no-sprint') {
        newTask = await Task.findByIdAndUpdate(req.params.taskId, {
sprint: null }, { new: true })
      } else {
        newTask = await Task.findByIdAndUpdate(req.params.taskId, {
sprint: req.body.sprint }, { new: true })
        await Sprint.findByIdAndUpdate(req.body.sprint ,  { $addToSet:
{ 'tasks': req.params.taskId } })
      }

      return res.json(newTask)
    } catch (err) {
      return res.status(400).json(err)
    }
  },
  async getTaskByName(req, res) {
    try {
      const tasks = await Task.find({'name': {'$regex':
req.params.name, '$options': 'i'}})
      return res.json(tasks)
    } catch (err) {
      return res.status(400).json(err)
    }
  }
}
```

UserController.js

```
const mongoose = require('mongoose')
const User = mongoose.model('User')
const error = require('../utils/errors')

module.exports = {
  async getAllUsers(req, res) {
    const { page = 1 } = req.query
    const users = await User.paginate({}, { page, limit: 10 })
    return res.json(users)
  },
  async getUserInfo(req, res) {
    const user = await User.findById(req.params.id).populate('tasks')

    return res.json(user)
  },
  async updateUser(req, res) {
    try {
      const user = await User.findByIdAndUpdate(req.params.id,
req.body, { new: true })
      return res.json(user)
    } catch (err) {
      const responseErr = error.crudUserErrors[err.name] ?
error.crudUserErrors[err.name] : err
      return res.status(400).json(responseErr)
    }
  },
  async getUserByName(req, res) {
    try {
      const users = await User.find({ $or: [{'name.first': {'$regex':
req.params.name, '$options': 'i'}}, {'name.last':{'$regex':
req.params.name, '$options': 'i'}}]})
      return res.json(users)
    } catch (err) {
      const responseErr = error.crudUserErrors[err.name] ?
error.crudUserErrors[err.name] : err
      return res.status(400).json(responseErr)
    }
  }
}
```

Interval.js

```
const mongoose = require('mongoose')
const mongoosePaginate = require('mongoose-paginate')

const IntervalSchema = new mongoose.Schema({
  clockIn: {
    type: Date,
    required: true
  },
  clockOut: {
    type: Date
  },
  task: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Task'
  },
  dayOrder: {
    type: Number,
    required: true
  }
}, { timestamps: true, strict: false })

IntervalSchema.plugin(mongoosePaginate)

mongoose.model('Interval', IntervalSchema)
```

Organization.js

```
const mongoose = require('mongoose')
const mongoosePaginate = require('mongoose-paginate')

const OrganizationSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    unique: true
  },
  description: {
    type: String
  },
  users: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  }],
```

```
  active: {
    type: Boolean,
    default: true
  }
}, { timestamps: true })

OrganizationSchema.plugin(mongoosePaginate)

mongoose.model('Organization', OrganizationSchema)
```

Project.js

```
const mongoose = require('mongoose')
const mongoosePaginate = require('mongoose-paginate')

const ProjectSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    unique: true
  },
  description: {
    type: String
  },
  organization: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Organization'
  },
  users: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  }],
  sprints: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Sprint'
  }],
  tasks: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Task'
  }],
  estimated: {
    type: Number,
  },
  completed: {
```

```
    type: Number
  },
  remain: {
    type: Number
  }
}, { timestamps: true })

ProjectSchema.plugin(mongoosePaginate)

mongoose.model('Project', ProjectSchema)
```

Sprint.js

```
const mongoose = require('mongoose')
const mongoosePaginate = require('mongoose-paginate')

const SprintSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true
  },
  number: {
    type: Number,
    required: true
  },
  project: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Project'
  },
  tasks: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Task'
  }],
  expectDate: {
    type: Date
  },
  active: {
    type: Boolean
  }
}, { timestamps: true })

SprintSchema.plugin(mongoosePaginate)

mongoose.model('Sprint', SprintSchema)
```

Task.js

```
const mongoose = require('mongoose')
const mongoosePaginate = require('mongoose-paginate')

const TaskSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true
  },
  number: {
    type: Number
  },
  description: {
    type: String
  },
  comments: [{
    type: String
  }],
  user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  },
  subtasks: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Task'
  }],
  estimated: {
    type: Number,
  },
  completed: {
    type: Number
  },
  remain: {
    type: Number
  },
  status: {
    type: String,
    enum: ['open', 'doing', 'committed', 'testing', 'done']
  },
  type: {
    type: String,
    enum: ['backlog', 'bug', 'refactor']
```

```
  },
  acceptanceCriteria: [{
    order: {
      type: Number
    },
    description: {
      type: String
    },
    checked: {
      type: Boolean
    },
    accepted: {
      type: Boolean
    }
  }],
  parentTask: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Task'
  },
  sprint: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Sprint'
  }
}, { timestamps: true })

TaskSchema.plugin(mongoosePaginate)

mongoose.model('Task', TaskSchema)
```

User.js

```
const mongoose = require('mongoose')
const mongoosePaginate = require('mongoose-paginate')

const UserSchema = new mongoose.Schema({
  login: {
    type: String,
    trim: true,
    required: true,
    unique: true
  },
  password: {
    type: String,
    trim: true,
```

```
    required: true
  },
  name: {
    first: {
      type: String,
      trim: true,
      required: true
    },
    last: {
      type: String,
      trim: true,
      required: true
    }
  },
  cpf: {
    type: String,
    required: true,
    unique: true,
    match: /^[0-9]*$/,
    minlength: 11,
    maxlength: 11
  },
  birthdate: {
    type: Date,
    required: true
  },
  reports: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'UserMonthReport'
  }],
  tasks: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Task'
  }],
  recoverPassword: {
    type: Number
  },
  organization: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Organization'
  }
}, { toJSON: { virtuals: true }, timestamps: true })
```

```
UserSchema.plugin(mongoosePaginate)

UserSchema.virtual('fullName').get(function () {
  return `${this.name.first} ${this.name.last}`;
});

mongoose.model('User', UserSchema)
```

UserMonthReport.js

```
const mongoose = require('mongoose')
const mongoosePaginate = require('mongoose-paginate')

const UserMonthReportSchema = new mongoose.Schema({
  monthOrder: {
    type: Number,
    enum: ['0','1', '2', '3', '4', '5', '6', '7', '8', '9', '10',
'11'],
    required: true
  },
  intervals: [{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Interval'
  }],
  monthHours: {
    type: Number,
    default: 0
  },
  year: {
    type: Number,
    required: true
  },
  user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User'
  }
}, { timestamps: true, strict: false })

UserMonthReportSchema.plugin(mongoosePaginate)

mongoose.model('UserMonthReport', UserMonthReportSchema)
```

emailService.js

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

```
exports.transporter = nodemailer.createTransport({
  host: "smtp.gmail.com",
  port: 587,
  secure: false, // true for 465, false for other ports
  auth: {
    user: "",
    pass: ""
  }
});
```

errors.js

```
exports.loginErrors = {
  NotFound: {
    message: 'Credenciais inválidas.'
  }
}

exports.recoverPasswordErrors = {
  NotFound: {
    message: 'E-mail não cadastrado.'
  },
  SendingError: {
    message: 'Erro ao recuperar senha, tente novamente.'
  },
  InvalidId: {
    message: 'Usuário inválido.'
  },
  ExpiredSolicitation: {
    message: 'Solicitação expirada, refaça a solicitação.'
  }
}

exports.crudUserErrors = {
  ValidationError: {
    message: 'Faltam informações do usuário.'
  },
  MongoError: {
    message: 'Usuário já cadastrado no sistema.',
    action: 'recoverPassword'
  }
}
```

routes.js

```
const express = require('express')
const routes = express.Router()

const UserController = require('./controllers/UserController')
const OpenController = require('./controllers/OpenController')
const AuthController = require('./controllers/AuthController')
const ReportController = require('./controllers/ReportController')
const SituationController =
require('./controllers/SituationController')
const TaskController = require('./controllers/TaskController')
const SprintController = require('./controllers/SprintController')
const ProjectController = require('./controllers/ProjectController')

//Rotas sem autenticação
routes.post('/login', OpenController.login)
routes.post('/user', OpenController.createUser)
routes.post('/recoverPassword', OpenController.recoverPassword)
routes.put('/resetPassword/:id', OpenController.resetPassword)

//Rotas com autenticação
routes.get('/users', AuthController.verifyAuth,
UserController.getAllUsers)
routes.get('/user/me', AuthController.verifyAuth)
routes.put('/user/:id', AuthController.verifyAuth,
UserController.updateUser)
routes.get('/users/:name', AuthController.verifyAuth,
UserController.getUserByName)
routes.get('/user-by-id/:id', AuthController.verifyAuth,
UserController.getUserInfo)

routes.get('/report/:userId&:month&:year', AuthController.verifyAuth,
ReportController.getMonthReport)
routes.put('/report/:monthId', AuthController.verifyAuth,
ReportController.updateUserHoursReport)
routes.get('/reports/:user', AuthController.verifyAuth,
ReportController.getUserReports)

routes.get('/situations', AuthController.verifyAuth,
SituationController.getAllSituations)
```

```
routes.post('/intervals', AuthController.verifyAuth,
ReportController.updateIntervals)

routes.get('/task/:taskId', AuthController.verifyAuth,
TaskController.getTaskById)
routes.get('/tasks/:projectId', AuthController.verifyAuth,
TaskController.getTasksFromProject)
routes.post('/task', AuthController.verifyAuth,
TaskController.createTask)
routes.put('/task/:taskId', AuthController.verifyAuth,
TaskController.updateTask)
routes.put('/removeTask/:taskId', AuthController.verifyAuth,
TaskController.removeTask)
routes.put('/add-user-task/:taskId', AuthController.verifyAuth,
TaskController.addUserToTask)
routes.put('/remove-user-task/:taskId', AuthController.verifyAuth,
TaskController.removeUserFromTask)
routes.put('/setTaskInterval/:taskId', AuthController.verifyAuth,
TaskController.setTaskToInterval)
routes.put('/removeTaskInterval/:taskId', AuthController.verifyAuth,
TaskController.removeTaskFromInterval)
routes.post('/subtask', AuthController.verifyAuth,
TaskController.createSubtask)
routes.put('/update-sprint-task/:taskId', AuthController.verifyAuth,
TaskController.updateTaskSprint)
routes.get('/tasks-by-name/:name', AuthController.verifyAuth,
TaskController.getTaskByName)

routes.get('/sprints/:projectId', AuthController.verifyAuth,
SprintController.getSprintsFromProject)
routes.get('/sprint/:projectId', AuthController.verifyAuth,
SprintController.getActiveSprint)
routes.post('/sprint', AuthController.verifyAuth,
SprintController.createSprint)
routes.put('/sprint/:sprintId', AuthController.verifyAuth,
SprintController.updateSprint)

routes.get('/projects/:organizationId', AuthController.verifyAuth,
ProjectController.getProjectsFromOrg)
routes.get('/project/:projectId', AuthController.verifyAuth,
ProjectController.getProjectDetails)
routes.post('/project', AuthController.verifyAuth,
ProjectController.createProject)
```

```
routes.put('/project/:projectId', AuthController.verifyAuth,
ProjectController.updateProject)
routes.put('/add-user-project/:projectId', AuthController.verifyAuth,
ProjectController.addUserProject)
routes.put('/remove-user-project/:projectId',
AuthController.verifyAuth, ProjectController.removeUserProject)
routes.post('/create-new-sprint/:projectId', AuthController.verifyAuth,
ProjectController.createNewSprint)
routes.get('/filter-task-by-sprint/:projectId&:sprintId',
AuthController.verifyAuth, ProjectController.filterTasksBySprint)



module.exports = routes
```

server.js

```
const express = require('express')
const cors = require('cors')
const mongoose = require('mongoose')
const requireDir = require('require-dir')
const dotenv = require('dotenv')
const moment = require('moment')

//Checando se é ambiente de produção e utilizando .env
if(process.env.NODE_ENV !== 'production') {
  dotenv.config()
}
//Iniciando App
const app = express()
app.use(express.json())
app.use(cors())

//Atualizando o locale do moment para formato pt-br
moment.locale('pt-br')

//Iniciando DB
mongoose.connect('mongodb://localhost:27017/tccserver', {
useNewUrlParser: true, useUnifiedTopology: true })

requireDir('./src/models')

//Rotas
app.use('/api', require('./src/routes'))
```

```
app.listen(3001)
```

Frontend

Auth.ts

```
import { ILogin } from '../../../interfaces/api'
import { API } from '../api.config'

export const loginReq = async (body: ILogin) => {
  return await API.post('/login', body);
}

export const fetchUserInfoReq = async () => {
  return await API.get('/user/me');
}
```

Project.ts

```
import { IProject } from '../../../interfaces/api'
import { API } from '../api.config'

export const createProjectReq = async (body: IProject) => {
  return await API.post('/project', body);
}

export const getProjectReq = async (projectId: String) => {
  return await API.get(`/project/${projectId}`);
}

export const getOrgProjectsReq = async (orgId: String) => {
  return await API.get(`/projects/${orgId}`);
}

export const updateProjectReq = async (projectId: String, body: any) =>
{
  return await API.put(`/project/${projectId}`, body);
}

export const addUserToProjectReq = async (projectId: String, body: any)
=> {
  return await API.put(`/add-user-project/${projectId}`, body);
```

```
}

export const removeUserFromProjectReq = async (projectId: String, body:
any) => {
  return await API.put(`/remove-user-project/${projectId}`, body);
}

export const createNewSprintReq = async (projectId: String) => {
  return await API.post(`/create-new-sprint/${projectId}`);
}

export const filterTasksBySprintReq = async (projectId: String,
sprintId: String) => {
  return await
API.get(`/filter-task-by-sprint/${projectId}&${sprintId}`);
}
```

Report.ts

```
import { API } from '../api.config'

export const updateIntervalsReq = async (values: any) => {
  return await API.post('/intervals', values);
}

export const getUserMonthlyReportReq = async (userId: any, month:
number, year: number) => {
  return await API.get(`/report/${userId}&${month}&${year}`);
}
```

Task.ts

```
import { API } from '../api.config'

export const createTaskReq = async (values: any) => {
  return await API.post('/task', values);
}

export const getTaskReq = async (taskId: String) => {
  return await API.get(`/task/${taskId}`);
}

export const updateTaskReq = async (taskId: String, values: any) => {
  return await API.put(`/task/${taskId}`, values);
}
```

```
export const addUserToTaskReq = async (taskId: String, values: any) =>
{
  return await API.put(`/add-user-task/${taskId}`, values);
}

export const removeUserFromTaskReq = async (taskId: String, values:
any) => {
  return await API.put(`/remove-user-task/${taskId}`, values);
}

export const updateTaskSprintReq = async (taskId: String, values: any)
=> {
  return await API.put(`/update-sprint-task/${taskId}`, values);
}

export const getTasksByNameReq = async (name: string) => {
  return await API.get(`/tasks-by-name/${name}`);
}
```

User.ts

```
import { API } from '../api.config'

export const getUsersByNameReq = async (name: string) => {
  return await API.get(`/users/${name}`);
}

export const getUserInfoReq = async (id: string) => {
  return await API.get(`/user-by-id/${id}`);
}
```

api.config.ts

```
import axios from 'axios';

export const API = axios.create({
  baseURL: process.env.REACT_APP_BASE_URL + 'api'
});
```

factory.ts

```
export const getTagColor = (type: string) => {
  switch (type) {
    case 'bug':
      return 'red'
```

```
    case 'refactor':
      return 'orange'
    case 'backlog':
      return 'blue'
    default:
      return 'gray'
  }
}

export const getStatusTagColor = (type: string) => {
  switch (type) {
    case 'open':
      return 'magenta'
    case 'doing':
      return 'blue'
    case 'committed':
      return 'purple'
    case 'testing':
      return 'cyan'
    case 'done':
      return 'green'
    default:
      return 'gray'
  }
}
```

pt.js

```
export const pt = {
  translation: {
    button: {
      send: 'Enviar',
      add: 'Adicionar',
      remove: 'Remover',
      manage: 'Gerenciar',
      cancel: 'Cancelar'
    },
    form: {
      userName: 'Usuário',
      password: 'Senha',
      reqLogin: 'Insira seu e-mail.',
      reqPassword: 'Insira sua senha.',
      invalidCredentials: 'Credenciais inválidas.'
    },
```

```
  footer: {
    message: 'Scout App ©2021 Created by Scout Inc'
  },
  nav: {
    home: 'Início',
    projects: 'Projetos',
    analytics: 'Gerenciamento',
    reports: 'Relatórios',
    logout: 'Sair',
    add: 'Adicionar',
    fetchProjectsError: 'Erro ao buscar projetos.'
  },
  home: {
    welcome: 'Bem vindo! Hoje é ',
    tasks: 'Essas são suas tarefas:',
    hours: 'Seus últimos lançamentos:',
    newDay: 'Inicie seu dia:',
    clockIn: 'Começar',
    clockOut: 'Pausar'
  },
  projects: {
    welcome: 'Projetos',
    add: 'Adicionar projeto',
    name: 'Nome:',
    description: 'Descrição:',
    estimated: 'Tempo estimado:',
    required: 'Por favor, insira um nome.',
    days: 'dias',
    success: 'Projeto criado com sucesso!',
    error400: 'Nome de projeto já utilizado!',
    cta: 'Adicionar',
    successFetch: 'Informações do projeto carregadas.',
    errorFetch: 'Erro ao buscar informações do projeto',
    edit: 'Editar',
    creationDate: 'Data de criação: ',
    users: 'Usuários do projeto',
    tasks: 'Tasks',
    modalTitle: 'Gerenciar projeto',
    updateProjectSuccess: 'Projeto atualizado com sucesso!',
    updateProjectError: 'Erro ao atualizar o projeto!',
    board: 'Quadro de acompanhamento'
  },
  general: {
```

```
      error404: 'Erro interno do servidor, tente novamente mais
tarde!',
      filters: 'Filtros:'
    },
    manageUser: {
      modalTitle: 'Gerenciar usuários',
      searchPlaceholder: 'Adicione usuários',
      notFound: 'Sem resultados...',
      addSuccess: 'Usuário adicionado com sucesso!',
      addError: 'Erro ao adicionar usuário!',
      userAlreadyIn: 'Usuário já está no projeto!',
      removeSuccess: 'Usuário removido com sucesso!',
      removeError: 'Erro ao remover usuário!',
    },
    manageTask: {
      modalTitle: 'Gerenciar task',
      form: {
        name: 'Nome',
        description: 'Descrição',
        user: 'Usuário responsável',
        subtasks: 'Sub-tasks',
        estimated: 'Tempo estimado',
        estimatedTooltip: 'Quantas horas necessárias para completar a
task?',
        type: 'Tipo da task',
        typePlaceholder: 'Selecione o tipo da task',
        typeBug: 'Bug',
        typeBacklog: 'Backlog',
        typeRefactor: 'Refatoração',
        accptanceCriteria: 'Critérios de aceite',
        sprint: 'Sprint',
        status: 'Status da task',
        statusPlaceholder: 'Selecione o status da task'
      },
      removeSprint: 'Sem sprint',
      createTaskSuccess: 'Task criada com sucesso!',
      createTaskError: 'Erro ao criar a task!',
      updateTaskSuccess: 'Task atualizada com sucesso!',
      updateTaskError: 'Erro ao atualizar a task!',
      removeUserSuccess: 'Usuário removido com sucesso!',
      removeUserError: 'Erro ao remover usuário!',
      addUserSuccess: 'Usuário adicionado com sucesso!',
      addUserError: 'Erro ao adicionar usuário!',
```

```
    updateSprintSuccess: 'Sprint atualizada com sucesso!',
    updateSprintError: 'Erro ao atualizar a sprint!',
    notFound: 'Sem resultados',
    searchPlaceholder: 'Busque pela task'
  },
  reports: {
    title: 'Relatório de horas',
    dayOrder: 'Dia',
    dayHours: 'Horas contabilizadas',
    intervals: 'Intervalos',
    monthHours: 'Banco de horas mensal:'
  },
  board: {
    title: 'Quadro de acompanhamento',
    open: 'Em aberto',
    doing: 'Em Andamento',
    testing: 'Em teste',
    done: 'Finalizado',
    createNewSprint: 'Criar nova sprint',
    createNewSprintSuccess: 'Nova sprint criada com sucesso!',
    createNewSprintError: 'Erro ao criar nova sprint!',
    backlog: 'Exibir backlog completo',
    sprintSelect: 'Escolha a sprint',
    fetchSprintError: 'Erro ao buscar informações da sprint!'
  },
  intervalTask: {
    modalTitle: 'Informações diárias:',
    intervals: 'Intervalos registrados:',
    addInterval: 'Adicionar novo intervalo',
    linkTask: 'Vincular task'
  },
  task: {
    name: 'Nome:',
    description: 'Descrição:',
    estimated: 'Tempo estimado:',
    completed: 'Tempo completado:',
    required: 'Por favor, insira um nome.',
    days: 'dias',
    success: 'Task criado com sucesso!',
    error400: 'Nome de task já utilizado!',
    cta: 'Adicionar',
    successFetch: 'Informações da task carregadas.',
    errorFetch: 'Erro ao buscar informações da task.',
```

```
      edit: 'Editar',
      creationDate: 'Data de criação: ',
      user: 'Usuário responsável:',
      status: 'Status da task:',
      sprint: 'Sprint: ',
      noSprint: 'Sem sprint definida'
    },
  }
}
```

TaskList.ts

```
import React, { FunctionComponent, useEffect, useState } from 'react';
import { Button, Divider, List, Tag } from 'antd';
import { useTranslation } from 'react-i18next';
import { useParams, useHistory } from "react-router-dom";
import { TaskListProps } from '../../interfaces/atoms'

export const TaskList = ({ tasks, loading }: TaskListProps) => {
  const { t } = useTranslation();
  const { id } = useParams<{ id: string }>();
  const history = useHistory();

  const getTagColor = (type: string) => {
    switch (type) {
      case 'bug':
        return 'red'
      case 'refactor':
        return 'orange'
      case 'backlog':
        return 'blue'
      default:
        return 'gray'
    }
  }

  return (
    <>
     { tasks &&
      <List
        grid={{ gutter: 16, column: 1 }}
        dataSource={tasks}
        renderItem={(item:any) => (
          <List.Item>
```

```
            <List.Item.Meta
              avatar={
                item.type && <Tag
color={getTagColor(item.type)}>{item.type.toUpperCase()}</Tag>
              }
              title={<a onClick={() =>
history.push(`/tasks/${item._id}`)}>#{item.number} - {item.name}</a>}
              description={`Estimado: ${item.estimated}`}
            />
            <Divider></Divider>
          </List.Item>
        )}
      />}
    </>
  );
}
```

TaskSearchInput.ts

```
import React, { FunctionComponent, useEffect, useState } from 'react';
import { AutoComplete, Input, message } from 'antd';
import { useTranslation } from 'react-i18next';
import { useParams, useHistory } from "react-router-dom";
import { ITaskSearchInput } from '../../interfaces/atoms';
import { getTasksByNameReq } from '../../api/rest/queries/task';
const { TextArea, Search } = Input;

export const TaskSearchInput = ({ handleSelectTask, loading,
searchTermOut }: ITaskSearchInput) => {
  const { t } = useTranslation();
  const { id } = useParams<{ id: string }>();
  const [taskOptions, setTaskOptions] = useState<any>([]);
  const [searchTerm, setSearchTerm] = useState<string |
undefined>(searchTermOut)

  const handleSearchUser = (searchTerm: string) => {
    if(searchTerm.length < 3) return

    getTasksByNameReq(searchTerm).then(response => {
      const filteredData = response.data.map((item: any) => {
        return { value: item._id, label: item.name }
      })
      setTaskOptions(filteredData);
```

```
    }).catch(err => {
      console.error(err);
      message.error(t('projects.errorFetch'));
    })
  }

  const handleSelect = (item: any, option: any) => {
    handleSelectTask(option);
    setSearchTerm(option.label)
  }


  return (
    <AutoComplete
      options={taskOptions}
      onSelect={handleSelect}
      onSearch={handleSearchUser}
      value={searchTerm}
      onChange={(value) => setSearchTerm(value)}
      notFoundContent={t('manageTask.notFound')}
      disabled={loading}
    >
      <Search placeholder={t('manageTask.searchPlaceholder')}
enterButton />
    </AutoComplete>
  );
}
```

UserList.ts

```
import React, { FunctionComponent, useEffect, useState } from 'react';
import { Button, List, Avatar, } from 'antd';
import { useTranslation } from 'react-i18next';
import { useParams, useHistory } from "react-router-dom";
import { UserListProps } from '../../interfaces/atoms'

export const UserList = ({ handleRemove, users, loading }:
UserListProps) => {
  const { t } = useTranslation();
  const { id } = useParams<{ id: string }>();

  const checkHandleRemove = (item: any) => {
    if(handleRemove) {
```

```
      return (<Button danger type="text" onClick={() =>
handleRemove(item)}>{ t('button.remove') }</Button>)
    }
    return null
  }

  return (
    <List
        itemLayout="horizontal"
        dataSource={users}
        loading={loading}
        renderItem={(item:any) => (
          <List.Item actions={[ checkHandleRemove(item) ]}>
            <List.Item.Meta
              avatar={<Avatar
src="https://zos.alipayobjects.com/rmsportal/ODTLcjxAfvqbxHnVXCYX.png"
/>}
              title={<span>{item.fullName}</span>}
              description={item.login}
            />
          </List.Item>
        )}
      />
  );
}
```

UserSearchInput.ts

```
import React, { FunctionComponent, useEffect, useState } from 'react';
import { AutoComplete, Input, message } from 'antd';
import { useTranslation } from 'react-i18next';
import { useParams, useHistory } from "react-router-dom";
import { IUserSearchInput } from '../../interfaces/atoms';
import { getUsersByNameReq } from '../../api/rest/queries/user';
const { TextArea, Search } = Input;

export const UserSearchInput = ({ handleSelectUser, loading }:
IUserSearchInput) => {
  const { t } = useTranslation();
  const { id } = useParams<{ id: string }>();
  const [userOptions, setUserOptions] = useState<any>([]);
  const [searchTerm, setSearchTerm] = useState<string>('')
```

```
  const handleSearchUser = (searchTerm: string) => {
    if(searchTerm.length < 3) return

    getUsersByNameReq(searchTerm).then(response => {
      const filteredData = response.data.map((item: any) => {
        return { value: item.id, label: item.fullName }
      })
      setUserOptions(filteredData);
    }).catch(err => {
      console.error(err);
      message.error(t('projects.errorFetch'));
    })
  }

  const handleSelect = (item: any) => {
    handleSelectUser(item);
    setSearchTerm('');
    setUserOptions([]);
  }


  return (
    <AutoComplete
      options={userOptions}
      onSelect={(item) => handleSelect(item)}
      onSearch={handleSearchUser}
      value={searchTerm}
      onChange={(value) => setSearchTerm(value)}
      notFoundContent={t('manageUser.notFound')}
      disabled={loading}
    >
      <Search size="large"
placeholder={t('manageUser.searchPlaceholder')} enterButton />
    </AutoComplete>
  );
}
```

ManageIntervalTaskModal.ts

```
import React, { useEffect, useState } from 'react';
import { Modal, message, Input, AutoComplete, Typography,Row, Tag,
Divider, Space, Button, TimePicker } from 'antd';
import { useTranslation } from 'react-i18next';
```

```
import { addUserToProjectReq, removeUserFromProjectReq } from
'../../../api/rest/queries/project';
import { getUsersByNameReq } from '../../../api/rest/queries/user';
import { useParams } from "react-router-dom";
import { IntervalTaskModalProps } from
'../../../interfaces/components';
import { UserList } from '../../../atoms/UserList';
import { TaskSearchInput } from '../../../atoms/TaskSearchInput'
import moment from 'moment';

const { Search } = Input
const { Title } = Typography

export const ManageIntervalTaskModal = ({ isVisible, handleOk,
handleCancel, interval }: IntervalTaskModalProps) => {
  const { t } = useTranslation();
  const { id } = useParams<{ id: string }>();
  const [infoInterval, setInfoInterval] = useState(interval)

  useEffect(() => {
    if(interval) {
      setInfoInterval(interval)
    }
  }, [interval])

  const handleAddInterval = () => {
    const newInfoInterval = {...infoInterval};
    const newInterval = {clockIn: moment().format(), clockOut:
moment().format()}
    newInfoInterval.intervals ?
newInfoInterval.intervals.push(newInterval) : newInfoInterval.intervals
= [newInterval]
    setInfoInterval(newInfoInterval)
  }

  const handleSelectTime = (time: any, index: any, type: string) => {
    const newIntervals = {...infoInterval}
    newIntervals.intervals[index][type] = time
    setInfoInterval(newIntervals)
  }

  const handleLinkTask = (task: any, index: any) => {
    const newIntervalsLink = {...infoInterval}
```

```
    newIntervalsLink.intervals[index].task = task
    setInfoInterval(newIntervalsLink)
  }

  return (
    <Modal width={650} title={`${t('intervalTask.modalTitle')}
${infoInterval.date}`} visible={isVisible} onOk={() =>
handleOk(infoInterval)} onCancel={handleCancel}>
      { infoInterval &&
        <>
          <Title level={5}>{ t('intervalTask.intervals') }</Title>
          { infoInterval.intervals && infoInterval.intervals.map((item:
any, index: any) => (
            <>
              <Row key={index}>
                <TimePicker value={moment(item.clockIn)}
format={'HH:mm'} onSelect={(time) => handleSelectTime(time, index,
'clockIn')} style={{marginRight: '5px'}}/>
                <TimePicker value={item.clockOut ?
moment(item.clockOut) : null} format={'HH:mm'} onSelect={(time) =>
handleSelectTime(time, index, 'clockOut')} style={{marginRight:
'10px'}}/>
                <TaskSearchInput searchTermOut={item.task ?
item.task.name : ''} handleSelectTask={(task) => handleLinkTask(task,
index)}/>
              </Row>
              <Divider></Divider>
            </>

          ))}
          <Button
onClick={handleAddInterval}>{t('intervalTask.addInterval')}</Button>
        </>
      }
    </Modal>
  );
}
```

ManageProjectModal.ts

```
import React, { useEffect, useState } from 'react';
import { Modal, Button, message, Input, Form } from 'antd';
import { useTranslation } from 'react-i18next';
```

```
import { updateProjectReq } from '../../../api/rest/queries/project';
import { ManageProjectModalProps } from
'../../../interfaces/components'
const { TextArea } = Input;


export const ManageProjectModal = ({ isVisible, handleOk, handleCancel,
project }: ManageProjectModalProps) => {
  const { t } = useTranslation();
  const [loading, setLoading] = useState<boolean>(false);
  const [ form ] = Form.useForm()

  useEffect(() => {
    form.setFieldsValue(project);
  }, [project])

  const validateMessages = {
    required: '${label} é obrigatório!'
  };

  const handleSaveProject = (values: any) => {
    if(values.name !== project.name || values.description !==
project.description) {
      setLoading(true);
      updateProjectReq(project._id, values).then(response => {
        message.success(t('projects.updateProjectSuccess'))
        handleOk(response.data);
      }).catch(err => {
        message.error(t('projects.updateProjectError'))
      }).finally(() => {
        setLoading(false);
      })
    }
  }

  return (
    <Modal title={t('projects.modalTitle')} visible={isVisible}
footer={false} onCancel={handleCancel}>
      <Form form={form} name="task-form" onFinish={handleSaveProject}
validateMessages={validateMessages}>

        <Form.Item name="name" label={t('manageTask.form.name')}
rules={[{ required: true }]}>
```

```
          <Input />
        </Form.Item>

        <Form.Item name="description"
label={t('manageTask.form.description')}>
          <TextArea />
        </Form.Item>

        <Form.Item style={{textAlign: 'end'}}>
          <Button htmlType="button" onClick={handleCancel}
disabled={loading}>
            { t('button.cancel') }
          </Button>
          <Button type="primary" htmlType="submit" style={{marginLeft:
'10px'}} disabled={loading}>
            { t('button.send') }
          </Button>
        </Form.Item>
      </Form>
    </Modal>
  );
}
```

ManageTaskModal.ts

```
import React, { useEffect, useState } from 'react';
import { Modal, Button, message, Input, Form, Avatar, InputNumber,
Select } from 'antd';
import { useTranslation } from 'react-i18next';
import { getUserInfoReq } from '../../../api/rest/queries/user';
import { createTaskReq, updateTaskReq, removeUserFromTaskReq,
addUserToTaskReq, updateTaskSprintReq } from
'../../../api/rest/queries/task';
import { useParams } from "react-router-dom";
import { ManageTaskModalProps } from '../../../interfaces/components'
import { UserSearchInput } from '../../../atoms/UserSearchInput'
import { MinusCircleOutlined, UserOutlined } from '@ant-design/icons';
import { TextWrapper } from './style';
const { TextArea } = Input;
const { Option } = Select;
```

```
export const ManageTaskModal = ({ isVisible, handleOk, handleCancel,
task, project }: ManageTaskModalProps) => {
  const { t } = useTranslation();
  const { id } = useParams<{ id: string }>();
  const [selectedUser, setSelectedUser] = useState<any>('')
  const [loading, setLoading] = useState<boolean>(false);
  const [ form ] = Form.useForm()

  const validateMessages = {
    required: '${label} é obrigatório!'
  };

  useEffect(() => {
    form.setFieldsValue(task);
    if(task && task.user) {
      setSelectedUser(task.user)
    }
    if(task && task.sprint) {
      form.setFieldsValue({sprint: task.sprint._id})
    }
  }, [task])

  const handleSaveTask = (values: any) => {
    if(task) {
      if(values.sprint === 'no-sprint') values.sprint = null
      handleEditTask(values);
    } else {
      handleCreateTask(values);
    }
  }

  const handleCreateTask = (values: any) => {
    const reqObj = {
      task: values,
      project: id
    }

    setLoading(true);
    createTaskReq(reqObj).then(response => {
      message.success(t('manageTask.createTaskSuccess'));
      setSelectedUser('');
      form.resetFields();
      handleOk()
```

```
    }).catch(err => {
      message.error(t('manageTask.createTaskError'))
    }).finally(() => {
      setLoading(false);
    })
  }

  const handleEditTask = (values: any) => {
    setLoading(true);
    updateTaskReq(task._id, values).then(response => {
      message.success(t('manageTask.updateTaskSuccess'));
      setSelectedUser('');
      form.resetFields();
      handleOk()
    }).catch(err => {
      message.error(t('manageTask.updateTaskError'))
    }).finally(() => {
      setLoading(false);
    })
  }

  const handleSelect = (user: any) => {
    if(task) {
      addUserToTaskReq(task._id, {user}).then(response => {
        setSelectedUser(response.data.user)
        message.success(t('manageTask.addUserSuccess'))
      }).catch(err => {
        message.error(t('manageTask.addUserError'))
      })
    } else {
      getUserInfoReq(user).then(response => {
        setSelectedUser(response.data)
        form.setFieldsValue({user: response.data.id})
      }).catch(err => {
        message.error(t('general.error'))
      })
    }
  }

  const handleRemoveUser = () => {
    if(task) {
      removeUserFromTaskReq(task._id, {user:
selectedUser.id}).then(response => {
```

```
        setSelectedUser('')
        form.setFieldsValue({user: null})
        message.success(t('manageTask.removeUserSuccess'));
      })
    } else {
      setSelectedUser('')
      form.setFieldsValue({user: null})
      message.error(t('manageTask.removeUserError'));
    }
  }

  const handleSelectSprint = (value: any) => {
    updateTaskSprintReq(task._id, {sprint: value}).then(response => {
      message.success(t('manageTask.updateSprintSuccess'));
    }).catch(err => {
      message.error(t('manageTask.updateSprintError'));
    })
  }

  return (
    <Modal title={t('manageTask.modalTitle')} visible={isVisible}
footer={false} onCancel={handleCancel}>
      <Form form={form} name="task-form" onFinish={handleSaveTask}
validateMessages={validateMessages}>

        <Form.Item name="name" label={t('manageTask.form.name')}
rules={[{ required: true }]}>
          <Input />
        </Form.Item>

        <Form.Item name="description"
label={t('manageTask.form.description')}>
          <TextArea />
        </Form.Item>

        <Form.Item name="sprint" label={t('manageTask.form.sprint')}>
          <Select placeholder={t('board.sprintSelect')}
onChange={handleSelectSprint} disabled={loading}>
            <Option
value="no-sprint">{t('manageTask.removeSprint')}</Option>
            { project && project.sprints && project.sprints.map((item:
any) => (
              <Option value={item._id}>{item.name}</Option>
```

```
            ))}
          </Select>
        </Form.Item>

        <Form.Item name="user" label={t('manageTask.form.user')}>
          { !selectedUser && <UserSearchInput
handleSelectUser={handleSelect}/> }
          { selectedUser &&
            <span className="user">
              <Avatar icon={<UserOutlined />} />
              <TextWrapper>{selectedUser.fullName}</TextWrapper>
              <MinusCircleOutlined onClick={() => handleRemoveUser()}
/>
            </span> }
        </Form.Item>

        <Form.Item
          name="estimated"
          label={t('manageTask.form.estimated')}
          tooltip={t('manageTask.form.estimatedTooltip')}
          rules={[{ required: true }]}
          initialValue={1}
        >
          <InputNumber min={1} max={99} />
        </Form.Item>

        <Form.Item
          name="type"
          label={t('manageTask.form.type')}
          rules={[{ required: true }]}
        >
          <Select placeholder={t('manageTask.form.typePlaceholder')}>
            <Option
value="backlog">{t('manageTask.form.typeBacklog')}</Option>
            <Option value="bug">{t('manageTask.form.typeBug')}</Option>
            <Option
value="refactor">{t('manageTask.form.typeRefactor')}</Option>
          </Select>
        </Form.Item>

        { task &&
          <Form.Item
            name="status"
```

```
            label={t('manageTask.form.status')}
            rules={[{ required: true }]}
          >
            <Select
placeholder={t('manageTask.form.statusPlaceholder')}>
              <Option value="open">{t('board.open')}</Option>
              <Option value="doing">{t('board.doing')}</Option>
              <Option value="testing">{t('board.testing')}</Option>
              <Option value="done">{t('board.done')}</Option>
            </Select>
          </Form.Item>
        }

        <Form.Item style={{textAlign: 'end'}}>
          <Button htmlType="button" onClick={handleCancel}
disabled={loading}>
            { t('button.cancel') }
          </Button>
          <Button type="primary" htmlType="submit" style={{marginLeft:
'10px'}} disabled={loading}>
            { t('button.send') }
          </Button>
        </Form.Item>
      </Form>
    </Modal>
  );
}
```

ManageUserModal.ts

```
import React, { useEffect, useState } from 'react';
import { Modal, message, Input, AutoComplete} from 'antd';
import { useTranslation } from 'react-i18next';
import { addUserToProjectReq, removeUserFromProjectReq } from
'../../../api/rest/queries/project';
import { getUsersByNameReq } from '../../../api/rest/queries/user';
import { useParams } from "react-router-dom";
import { ManageUserModalProps } from '../../../interfaces/components';
import { UserList } from '../../../atoms/UserList';
import { UserSearchInput } from '../../../atoms/UserSearchInput'

const { Search } = Input
```

```
export const ManageUserModal = ({ isVisible, handleOk, handleCancel,
users }: ManageUserModalProps) => {
  const { t } = useTranslation();
  const { id } = useParams<{ id: string }>();
  const [userArray, setUserArray] = useState<any>(users || []);
  const [loading, setLoading] = useState<boolean>(false);
  const [updatedProject, setUpatedProject] = useState<any>({})

  useEffect(() => {
    if(users) {
      setUserArray(users)
    }
  }, [users])

  const handleSelectUser = (item: any) => {
    setLoading(true);

    addUserToProjectReq(id, { userId: item }).then(response => {

      if(response.data.users.length === userArray.length) {
        return message.warning(t('manageUser.userAlreadyIn'))
      }

      setUserArray(response.data.users);
      setUpatedProject(response.data);

      message.success(t('manageUser.addSuccess'));
    }).catch(err => {
      message.error(t('manageUser.addError'));
    }).finally(() => {
      setLoading(false);
    })
  }

  const handleRemoveUser = (item: any) => {
    setLoading(true);

    removeUserFromProjectReq(id, {userId: item._id}).then(response => {

      setUserArray(response.data.users);
      setUpatedProject(response.data);

      message.success(t('manageUser.removeSuccess'));
```

```
    }).catch(err => {
      message.error(t('manageUser.removeError'));
    }).finally(() => {
      setLoading(false)
    })
  }

  return (
    <Modal title={t('manageUser.modalTitle')} visible={isVisible}
onOk={() => handleOk(updatedProject)} onCancel={handleCancel}>
      <UserSearchInput handleSelectUser={handleSelectUser}
loading={loading}/>
      <UserList users={userArray} handleRemove={handleRemoveUser}
loading={loading}/>
    </Modal>
  );
}
```

Navigation.ts

```
import React, { FunctionComponent, useState, useEffect } from 'react';
import { AppRouter } from '../../routes/Router';
import { Layout, Menu, message } from 'antd';
import { LogoNav } from '../../style/global';
import logoNav from '../../assets/brand/logo.svg';
import { useTranslation } from 'react-i18next';
import {
  HomeOutlined,
  PieChartOutlined,
  FileSearchOutlined,
  LoginOutlined,
  PartitionOutlined,
  PlusOutlined
} from '@ant-design/icons';
import { useSelector } from 'react-redux';
import {
  selectToken,
  logout,
  selectUserInfo,
  selectApiReady
} from '../../store/appSlice';
import { useDispatch } from 'react-redux';
import { useHistory } from "react-router-dom";
```

```
import { getOrgProjectsReq } from '../../api/rest/queries/project';
import { getUserInfoReq } from '../../api/rest/queries/user';
import { updateUserInfo } from '../../store/appSlice';

const { Header, Content, Sider } = Layout;
const { SubMenu } = Menu;

export const Navigation: FunctionComponent = () => {
  const [collapsed, setCollapsed] = useState(true);
  const [projects, setProjects] = useState([]);
  const token = useSelector(selectToken);
  const { t } = useTranslation();
  const dispatch = useDispatch();
  const history = useHistory();
  const userInfo = useSelector(selectUserInfo);
  const apiReady = useSelector(selectApiReady);

  const getOrgProjects = () => {
    const id = userInfo.organization || userInfo._id;

    getOrgProjectsReq(id).then(response => {
      setProjects(response.data.docs);
    }).catch(err => {
      message.error(t('nav.fetchProjectsError'));
    })
  }

  useEffect(() => {
    if(!projects.length && apiReady) {
      fetchUserInfo();
      getOrgProjects();
    }
  }, [apiReady])

  const fetchUserInfo = () => {
    getUserInfoReq(userInfo.id).then(response => {
      dispatch(updateUserInfo(response.data));
    })
  }

  const handleLogout = () => {
    dispatch(logout())
    history.push('/');
```

```
  }

  const handleMenuClick = (values: any) => {
    history.push(values.key);
  }

  return (
    <>
      { token &&
        <Sider collapsible collapsed={collapsed} onCollapse={() =>
setCollapsed(!collapsed)}>
          <LogoNav src={logoNav}/>
          <Menu theme="dark" defaultSelectedKeys={['1']} mode="inline"
onClick={handleMenuClick}>
            <Menu.Item key="/home" icon={<HomeOutlined />}>
              {t('nav.home')}
            </Menu.Item>
            <SubMenu key="sub1" icon={<PartitionOutlined />}
title={t('nav.projects')}>
              { projects.map((project: any) => (
                <Menu.Item key={`/projects/${project._id}`}>{
project.name }</Menu.Item>
              ))}
              <Menu.Item key="/projects/add"
icon={<PlusOutlined/>}>{t('nav.add')}</Menu.Item>
            </SubMenu>
            <Menu.Item key="/reports" icon={<FileSearchOutlined />}>
              {t('nav.reports')}
            </Menu.Item>
            <Menu.Item key="/" icon={<LoginOutlined />}
onClick={handleLogout}>
              {t('nav.logout')}
            </Menu.Item>
          </Menu>
        </Sider>
      }
      <Layout className="site-layout">
        <Header style={{ padding: 0 }}/>
        <Content style={{ margin: '0 16px' }}>
          <AppRouter />
        </Content>
      </Layout>
    </>
```

```
  );
}
```

ProjectDisplay.ts

```
import React, { FunctionComponent, useEffect, useState } from 'react';
import { PageWrapper } from '../../../style/global';
import { Card, Typography, message, Button, PageHeader, Divider, Row }
from 'antd';
import { useTranslation } from 'react-i18next';
import { getProjectReq } from '../../../api/rest/queries/project';
import { useParams, useHistory } from "react-router-dom";
import { useSelector } from 'react-redux';
import { selectApiReady } from '../../../store/appSlice';
import { ManageUserModal } from '../../Modals/ManageUserModal';
import { ManageTaskModal } from '../../Modals/ManageTaskModal';
import { ManageProjectModal } from '../../Modals/ManageProjectModal'
import { UserList } from '../../../atoms/UserList'
import { TaskList } from '../../../atoms/TaskList'

const { Title, Paragraph } = Typography;

export const DisplayProject: FunctionComponent = () => {
  const { t } = useTranslation();
  const { id } = useParams<{ id: string }>();
  const apiReady = useSelector(selectApiReady);
  const history = useHistory();
  const [project, setProject] = useState<any>({});
  const [loading, setLoading] = useState<boolean>(false);
  const [manageUserModal, setManageUserModal] =
useState<boolean>(false);
  const [manageTaskModal, setManageTaskModal] =
useState<boolean>(false);
  const [manageProjectModal, setManageProjectModal] =
useState<boolean>(false);

  useEffect(() => {
    if(project._id !== id && apiReady && !loading) {
      getProjectInfo();
    }
  }, [id])

  const getProjectInfo = () => {
    setLoading(true);
```

```
    getProjectReq(id).then(response => {
      setProject(response.data);
    }).catch(err => {
      console.error(err);
      message.error(t('projects.errorFetch'));
    }).finally(() => {
      setLoading(false);
    })
  }

  const handleManageUserOk = (projectUpdated: any) => {
    setProject(projectUpdated);
    setManageUserModal(false);
  }

  const handleManageTaskOk = () => {
    setManageTaskModal(false);
    getProjectInfo();
  }

  const handleManageProjectOk = (upatedProject: any) => {
    setManageProjectModal(false);
    setProject(upatedProject);
  }

  return (
    <PageWrapper>
      <PageHeader
        ghost={false}
        onBack={() => history.push('/projects')}
        title={ project.name }
        extra={[
          <Button key="board" onClick={() =>
history.push(`/board/${project._id}`)}>{ t('projects.board')
}</Button>,
          <Button key="edit" type="primary" onClick={() =>
setManageProjectModal(true)}>{ t('projects.edit') }</Button>,
        ]}
      />
      <Card>
        { project.description && <>
          <Paragraph>{ project.description }</Paragraph>
          <Divider/>
```

```
        </>}
        <Title level={5}>{ `${t('projects.creationDate')}
${project.createdAt}` }</Title>
        <Divider/>
        <Row>
          <Title style={{paddingRight: '10px'}} level={4}>{
t('projects.users') }</Title>
          <Button onClick={() => setManageUserModal(true)}>{
t('button.manage') }</Button>
        </Row>
        <UserList users={project.users} />
        <Divider/>
        <Row>
          <Title style={{paddingRight: '10px'}} level={4}>{
t('projects.tasks') }</Title>
          <Button onClick={() => setManageTaskModal(true)}>{
t('button.add') }</Button>
        </Row>

        <TaskList tasks={project.tasks}/>
      </Card>
      <ManageUserModal users={project.users}
isVisible={manageUserModal} handleOk={handleManageUserOk}
handleCancel={() => setManageUserModal(false)}/>
      <ManageTaskModal isVisible={manageTaskModal}
handleOk={handleManageTaskOk} handleCancel={() =>
setManageTaskModal(false)} project={project}/>
      <ManageProjectModal project={project}
isVisible={manageProjectModal} handleOk={handleManageProjectOk}
handleCancel={() => setManageProjectModal(false)}/>
    </PageWrapper>
  );
}
```

ProjectForm.ts

```
import React, { FunctionComponent, useState } from 'react';
import { PageWrapper } from '../../../style/global';
import { Form, Input, Button, Card, Typography, InputNumber, message }
from 'antd';
import { useTranslation } from 'react-i18next';
import { createProjectReq } from '../../../api/rest/queries/project';
import { useHistory } from "react-router-dom";
```

```
import { useSelector } from 'react-redux';
import { selectUserInfo } from '../../../store/appSlice';

const { Title } = Typography;

export const FormProject: FunctionComponent = () => {
  const [loading, setLoading] = useState(false);
  const { t } = useTranslation();
  const history = useHistory();
  const userInfo = useSelector(selectUserInfo)

  const [projectForm] = Form.useForm();

  const onFinish = (values: any) => {
    if(values.name) {
      setLoading(true);

      values.organization = userInfo.organization || userInfo._id;

      createProjectReq(values).then(response => {
        message.success(t('projects.success'))
        history.push(`/projects/${response.data._id}`);
        projectForm.resetFields();
      }).catch(err => {
        if(err.response.status === 400) {
          message.error(t('projects.error400'))
        } else {
          message.error(t('general.error404'))
        }
      }).finally(() => {
        setLoading(false);
      })
    }
  };

  return (
    <PageWrapper>
      <Title>{ t('projects.add')}</Title>
      <Card>
        <Form
          name="projectForm"
          onFinish={onFinish}
          form={projectForm}>
```

```
          <Form.Item
            label={ t('projects.name')}
            name="name"
            rules={[{ required: true, message:
t('projects.required')}]}
          >
            <Input />
          </Form.Item>

          <Form.Item
            label={ t('projects.description')}
            name="description"
          >
            <Input />
          </Form.Item>

          <Form.Item
            label={ t('projects.estimated')}
            name="estimated"
          >
            <InputNumber
              min={1}
              max={999}/>
              <span className="ant-form-text"> {
t('projects.days')}</span>
          </Form.Item>

          <Form.Item style={{ textAlign: 'end' }}>
            <Button type="primary" htmlType="submit" loading={loading}>
            { t('projects.cta')}
            </Button>
          </Form.Item>
        </Form>
      </Card>
    </PageWrapper>
  );
}
```

SearchInput.ts

```
import React, { useState, useCallback, FunctionComponent } from
'react';
import debounce from 'lodash.debounce';
```

```
import { Input } from 'antd';
import { Wrapper } from './style';

export const SearchInput: FunctionComponent<SearchInputProps> =
({placeholder, request}) => {
  const [inputValue, setInputValue] = useState('');

  const debouncedSave = useCallback(
    debounce((newValue: string) => request(newValue), 1000),
    []
  );

  const updateValue = (newValue: string) => {
    setInputValue(newValue);
    debouncedSave(newValue);
  }

  return (
    <Wrapper>
      <Input
        name='search-input'
        value={inputValue}
        onChange={input => updateValue(input.target.value)}
        placeholder={placeholder}/>
    </Wrapper>
  )
}

interface SearchInputProps {
  placeholder?: string,
  request: (arg: string) => void
}
```

TaskDisplay.ts

```
import React, { FunctionComponent, useEffect, useState } from 'react';
import { PageWrapper } from '../../../style/global';
import { Card, Typography, message, Button, PageHeader, Divider, Row,
Avatar, Tag } from 'antd';
import { useTranslation } from 'react-i18next';
import { getTaskReq } from '../../../api/rest/queries/task';
import { useParams, useHistory } from "react-router-dom";
import { useSelector } from 'react-redux';
import { selectApiReady } from '../../../store/appSlice';
```

```
import { ManageTaskModal } from '../../Modals/ManageTaskModal';
import { MinusCircleOutlined, UserOutlined } from '@ant-design/icons';
import { TextWrapper } from './style'
import { getTagColor, getStatusTagColor } from
'../../../assets/utils/factory'

const { Title, Paragraph } = Typography;

export const DisplayTask: FunctionComponent = () => {
  const { t } = useTranslation();
  const { id } = useParams<{ id: string }>();
  const apiReady = useSelector(selectApiReady);
  const history = useHistory();
  const [task, setTask] = useState<any>({});
  const [project, setProject] = useState<any>({});
  const [loading, setLoading] = useState<boolean>(false);
  const [manageTaskModal, setManageTaskModal] =
useState<boolean>(false);

  useEffect(() => {
    if(task._id !== id && apiReady && !loading) {
      getTaskInfo();
    }
  }, [id])

  const getTaskInfo = () => {
    setLoading(true);
    getTaskReq(id).then(response => {
      setTask(response.data.task);
      setProject(response.data.project);
    }).catch(err => {
      console.error(err);
      message.error(t('projects.errorFetch'));
    }).finally(() => {
      setLoading(false);
    })
  }

  const handleManageTaskOk = () => {
    setManageTaskModal(false);
    getTaskInfo();
  }
```

```
  return (
    <PageWrapper>
      <PageHeader
        ghost={false}
        onBack={() => history.goBack()}
        title={ [`#${task.number} - ${task.name}  `,
          <Tag color={getTagColor(task.type)}>{task.type &&
task.type.toUpperCase()}</Tag>,
          <Tag color={getStatusTagColor(task.status)}>{task.status &&
task.status.toUpperCase()}</Tag>
        ]}
        extra={[
          <Button key="edit" type="primary" onClick={() =>
setManageTaskModal(true)}>{ t('projects.edit') }</Button>,
        ]}
      />
      <Card>
        { task.description && <>
          <Paragraph>{ task.description }</Paragraph>
          <Divider/>
        </>}
        <Title level={5}>{ `${t('task.sprint')}` } {task.sprint &&
`${task.sprint.name}` || t('task.noSprint') }</Title>
        <Title level={5}>{ `${t('task.creationDate')}
${task.createdAt}` }</Title>
        <Title level={5}> {t('task.user')} { task.user &&
            <span className="user">
              <Avatar icon={<UserOutlined />} />
              <TextWrapper>{task.user.fullName}</TextWrapper>
            </span> }
        </Title>
        <Divider/>
        <Title level={5}> {t('task.estimated')} { task.estimated } {
t('task.days') }</Title>
        <Title level={5}> {t('task.completed')} { task.completed || '0'
} { t('task.days') }</Title>
        <Divider/>
      </Card>
      <ManageTaskModal task={task} isVisible={manageTaskModal}
handleOk={handleManageTaskOk} handleCancel={() =>
setManageTaskModal(false)} project={project}/>
    </PageWrapper>
  );
```

```
}
```

api.ts

```
export interface ILogin {
  login: String,
  password: String
}

export interface IProject {
  name: String,
  description?: String,
  estimated?: Number
}

export interface IResponse {
  error?: String,
  data?: String
}
```

atoms.ts

```
export interface UserListProps {
  handleRemove?: (item: any) => void;
  users: [];
  loading?: boolean;
}

export interface TaskListProps {
  handleRemove?: (item: any) => void;
  tasks: [];
  loading?: boolean;
}

export interface IUserSearchInput {
  handleRemove?: (item: any) => void;
  handleSelectUser: (item: any) => void;
  loading?: boolean;
}

export interface ITaskSearchInput {
  handleRemove?: (item: any) => void;
  handleSelectTask: (item: any) => void;
  loading?: boolean;
```

```
  searchTermOut?: string
}
```

components.ts

```
export interface ManageUserModalProps {
  isVisible: boolean;
  handleOk: (project: any) => void;
  handleCancel: () => void;
  users: [];
}

export interface ManageTaskModalProps {
  isVisible: boolean;
  handleOk: () => void;
  handleCancel: () => void;
  task?: any,
  project: any
}

export interface ManageProjectModalProps {
  isVisible: boolean;
  handleOk: (args: any) => void;
  handleCancel: () => void;
  project?: any
}

export interface IntervalTaskModalProps {
  isVisible: boolean;
  handleOk: (report: any) => void;
  handleCancel: () => void;
  interval?: any
}
```

AuthPage.ts

```
import React, { FunctionComponent, useState } from 'react';
import { LogoImage, PageWrapper } from '../../style/global';
import { Form, Input, Button, Card, Layout } from 'antd';
import { UserOutlined, LockOutlined } from '@ant-design/icons';
import logo from '../../assets/brand/logoName.svg';
import { CardWrapper } from './style';
import { useTranslation } from 'react-i18next';
import { useDispatch } from 'react-redux';
import { login } from '../../store/appSlice';
```

```
import { loginReq } from '../../api/rest/queries/auth';
import { ILogin } from '../../interfaces/api';
import { ValidateStatus } from 'antd/lib/form/FormItem';
const { Footer } = Layout;

export const Auth: FunctionComponent = () => {
  const { t } = useTranslation();
  const dispatch = useDispatch();
  const [error, setError] = useState<ValidateStatus>();
  const [errorMsg, setErrorMsg] = useState<String>();

  const onFinish = (values: ILogin) => {
    if(values.login  && values.password) {
      loginReq(values).then(response => {
        dispatch(login(response.data));
      }).catch(err => {
        setError('error');
        setErrorMsg(t('form.invalidCredentials'));
      })
    }
  };

  return (
    <PageWrapper>
      <CardWrapper>
        <Card style={{ width: '300px' }} cover={<LogoImage alt="logo"
src={logo} />}>
          <Form
            name="login"
            onFinish={onFinish}
          >
            <Form.Item
              name="login"
              validateStatus={error}
              rules={[{ required: true, message: t('form.reqLogin')
}]}>
              <Input prefix={<UserOutlined
className="site-form-item-icon" />} placeholder={ t('form.userName')}
/>
            </Form.Item>

            <Form.Item
              name="password"
```

```
              validateStatus={error}
              help={errorMsg}
              rules={[{ required: true, message: t('form.reqPassword')
}]}>
              <Input
                prefix={<LockOutlined className="site-form-item-icon"
/>}
                type="password"
                placeholder={ t('form.password')}
              />
            </Form.Item>

            <Form.Item style={{ textAlign: 'end' }}>
              <Button type="primary" htmlType="submit">
                {t('button.send')}
              </Button>
            </Form.Item>
          </Form>
        </Card>
      </CardWrapper>
      <Footer style={{ textAlign: 'center' }}>{
t('footer.message')}</Footer>
    </PageWrapper>
  );
}
```

BoardPage.ts

```
import React, { FunctionComponent, useEffect, useState } from 'react';
import { PageWrapper } from '../../style/global';
import { Typography, Card, message, PageHeader, Button, Select, Row }
from 'antd';
import { useTranslation } from 'react-i18next';
import { BoardWrapper } from './style';
import { useParams, useHistory } from "react-router-dom";
import { getProjectReq, createNewSprintReq, filterTasksBySprintReq }
from '../../api/rest/queries/project';
import { TaskList } from '../../atoms/TaskList';

const { Title } = Typography;
const { Option } = Select;

export const BoardDisplay: FunctionComponent = () => {
```

```
  const { t } = useTranslation();
  const { id } = useParams<{ id: string }>();
  const [project, setProject] = useState<any>({});
  const [loading, setLoading] = useState<boolean>(false);
  const [openTasks, setOpenTasks] = useState<any>([]);
  const [doingTasks, setDoingTasks] = useState<any>([]);
  const [testingTasks, setTestingTasks] = useState<any>([]);
  const [doneTasks, setDoneTasks] = useState<any>([]);
  const [selectedSprint, setSelectedSprint] = useState<any>('backlog');
  const history = useHistory();


  useEffect(() => {
    if(project._id !== id) {
      getProjectInfo();
    }
  }, [id])

  const getProjectInfo = () => {
    setLoading(true);
    getProjectReq(id).then(response => {
      setProject(response.data);
      if(!openTasks.length && !doingTasks.length &&
!testingTasks.length && !doneTasks.length) {
        prepareTasks(response.data.tasks);
      }
    }).catch(err => {
      console.error(err);
    }).finally(() => {
      setLoading(false);
    })
  }

  const prepareTasks = (tasks: any) => {
    for(let index = 0; index < tasks.length; index++) {

      if(tasks[index].status === 'open') {
        let newOpenTasks = openTasks;
        newOpenTasks.push(tasks[index])
        setOpenTasks(newOpenTasks)
      } else if(tasks[index].status === 'doing') {
        let newDoingTasks = doingTasks;
        newDoingTasks.push(tasks[index])
```

```
        setDoingTasks(newDoingTasks)
      } else if(tasks[index].status === 'testing') {
        let newTestingTasks = testingTasks;
        newTestingTasks.push(tasks[index])
        setTestingTasks(newTestingTasks)
      } else if(tasks[index].status === 'done') {
        let newDoneTasks = doneTasks;
        newDoneTasks.push(tasks[index])
        setDoneTasks(newDoneTasks)
      }
    }
  }

  const clearTasks = () => {
    setOpenTasks([]);
    setDoingTasks([]);
    setTestingTasks([]);
    setDoneTasks([]);
  }

  const handleCreateNewSprint = () => {
    createNewSprintReq(project._id).then(response => {
      setProject(response.data);

setSelectedSprint(response.data.sprints[response.data.sprints.length -
1]._id);
      clearTasks();
      message.success(t('board.createNewSprintSuccess'))
    }).catch(err => {
      message.error(t('board.createNewSprintError'))
    })
  }

  const handleSelect = (value: any) => {
    clearTasks();
    setSelectedSprint(value);

    if(value === 'backlog') {
      getProjectInfo()
    } else {
      filterTasksBySprintReq(project._id, value).then(response => {
        prepareTasks(response.data.tasks)
      }).catch(err => {
```

```
        message.error(t('board.fetchSprintError'))
      })
    }
  }

  return (
    <PageWrapper>
      <PageHeader
        ghost={false}
        onBack={() => history.goBack()}
        title={ `${t('board.title')} - ${project.name}` }
        extra={[
          <Button key="board" onClick={handleCreateNewSprint}>{
t('board.createNewSprint') }</Button>,
        ]}
      />
      <Card>
        <Row>
          <Title level={5} style={{marginRight: '10px'}}>{
t('general.filters')}</Title>
          <Select value={selectedSprint}
placeholder={t('board.sprintSelect')} onChange={handleSelect}
disabled={loading}>
            <Option value="backlog"
selected>{t('board.backlog')}</Option>
            { project.sprints && project.sprints.map((item: any) => (
              <Option value={item._id}>{item.name}</Option>
            ))}
          </Select>

        </Row>
      </Card>
      <BoardWrapper>
        <Card title={t('board.open')} style={{ width: '25%'}}>
          <TaskList tasks={openTasks} />
        </Card>
        <Card title={t('board.doing')} style={{ width: '25%'}}>
          <TaskList tasks={doingTasks} />
        </Card>
        <Card title={t('board.testing')} style={{ width: '25%'}}>
          <TaskList tasks={testingTasks} />
        </Card>
        <Card title={t('board.done')} style={{ width: '25%'}}>
```

```
          <TaskList tasks={doneTasks} />
        </Card>
      </BoardWrapper>
    </PageWrapper>
  );
}
```

HomePage.ts

```
import React, { FunctionComponent, useState, useEffect } from 'react';
import { PageWrapper } from '../../style/global';
import { Typography, Card, List, Button, Row, Tag } from 'antd';
import moment from 'moment';
import { useTranslation } from 'react-i18next';
import { getUserMonthlyReportReq } from '../../api/rest/queries/report'
import { useSelector } from 'react-redux';
import {
  selectUserInfo,
} from '../../store/appSlice';
import { TaskList } from '../../atoms/TaskList'

const { Title } = Typography;

export const Home: FunctionComponent = () => {
  const [todayDate, setTodayDate] = useState(moment().format('LLLL'));
  const userInfo = useSelector(selectUserInfo);
  const [userIntervals, setUserIntervals] = useState()
  const { t } = useTranslation();

  setInterval(() => {
    setTodayDate(moment().format('LLLL'));
  }, 10000)

  useEffect(() => {
    if(userInfo._id) {
      getUserMonthlyReportReq(userInfo._id, moment().month(),
moment().year()).then(response => {
        const sortedData = response.data.intervals.sort((a: any, b:
any) => {
          return b.dayOrder - a.dayOrder
        })

        const slicedData = sortedData.slice(0, 4)
```

```
        const formattedData = slicedData.map((item: any) => {
          return {...item, date: moment([response.data.year,
response.data.monthOrder, item.dayOrder]).format('L')}
        })

        setUserIntervals(formattedData)
      })
    }
  }, [userInfo])

  return (
    <PageWrapper>
      <Title>{ t('home.welcome')}{todayDate}.</Title>

      <Card>
        <Title level={4}>{ t('home.hours')}</Title>
        <List
          grid={{ gutter: 16, column: 4 }}
          dataSource={userIntervals}
          renderItem={(item:any) => (
            <List.Item>
              <List.Item.Meta
                title={<span>{moment(item.clockIn).format('HH:mm')} -
{moment(item.clockOut).format('HH:mm')}</span>}
                description={item.date}
              />
              {item.task && <span>{item.task.name}</span>}
            </List.Item>
          )}
        />
      </Card>
      <Card>
        <Title level={4}>{ t('home.tasks')}</Title>
        { userInfo && <TaskList tasks={userInfo.tasks}/> }
      </Card>
    </PageWrapper>
  );
}
```

ReportsPage.ts

```
import React, { FunctionComponent, useState, useEffect } from 'react';
```

```tsx
import { PageWrapper } from '../../style/global';
import { Typography, Card, Table, Button, Row, Tag, Calendar, message }
from 'antd';
import { useTranslation } from 'react-i18next';
import { ListWrapper } from './style';
import { updateIntervalsReq, getUserMonthlyReportReq } from
'../../api/rest/queries/report'
import { ManageIntervalTaskModal } from
'../../components/Modals/ManageIntervalTaskModal';
import { useSelector } from 'react-redux';
import {
  selectUserInfo,
} from '../../store/appSlice';
import moment from 'moment';
import 'moment/locale/pt-br'
moment.locale('pt-br');

const { Title } = Typography;

export const Reports: FunctionComponent = () => {
  const { t } = useTranslation();
  const [intervalTaskModal, setIntervalTaskModal] =
useState<boolean>(false);
  const [selectedReport, setSelectedReport] = useState<any>({});
  const [monthlyData, setMonthlyData] = useState<any>([])
  const userInfo = useSelector(selectUserInfo);

  useEffect(() => {
    if(userInfo._id) {
      getUserMonthlyReportReq(userInfo._id, moment().month(),
moment().year()).then(response => {
        formatData(response.data.intervals)
      })
    }
  }, [userInfo])

  const formatData = (rawData: any) => {
    const sortedData = rawData.sort((a: any, b: any) => {
      return a.dayOrder - b.dayOrder
    })

    let monthData = []
    for(let index = 0; index < sortedData.length; index++) {
```

```
      if(monthData.length === 0) {
        monthData[0] = {
          dayOrder: sortedData[index].dayOrder,
          intervals: [sortedData[index]]
        }
      } else {
        if(monthData[monthData.length - 1].dayOrder ===
sortedData[index].dayOrder) {
          monthData[monthData.length -
1].intervals.push(sortedData[index])
        } else {
          monthData.push({
            dayOrder: sortedData[index].dayOrder,
            intervals: [sortedData[index]]
          })
        }
      }
    }

    setMonthlyData(monthData);
  }

  const handleClickDay = (value: any) => {
    let selectedDay = monthlyData.find((item: any) => item.dayOrder ===
value.date());

    if(!selectedDay) {
      selectedDay = {
        dayOrder: value.date(),
        intervals: []
      }
    }

    selectedDay.date = value.format('L');
    selectedDay.originalDate = value;

    setSelectedReport(selectedDay);
    setIntervalTaskModal(true);
  }

  const handleSaveReport = (report: any) => {
    report.user = userInfo;
```

```
    updateIntervalsReq(report).then(response => {
      formatData(response.data.intervals)
      setIntervalTaskModal(false);
      message.success('Intervalo atualizado com sucesso!')
    }).catch(err => {

    })
  }

  function dateCellRender(value: any) {
    let listData:any = [];

    for(let index = 0; index < monthlyData.length; index++) {
      if(monthlyData[index].dayOrder === value.date()) {
        listData = monthlyData[index].intervals
        break
      }
    }

    return (
      <ListWrapper>
        {listData.map((item: any) => (
          <li key={item._id}>
            <Tag color={ item.clockIn && item.clockOut ? 'green' :
'red' }>
              {moment(item.clockIn).format('HH:mm')} -
{moment(item.clockOut).format('HH:mm')}
            </Tag>
          </li>
        ))}
      </ListWrapper>
    );
  }

  return (
    <PageWrapper>
      <Card>
        <Title level={4}>{ t('reports.title')}</Title>
        <Calendar dateCellRender={dateCellRender}
onSelect={handleClickDay}/>,
      </Card>
```

```
      <ManageIntervalTaskModal isVisible={intervalTaskModal}
handleOk={handleSaveReport} handleCancel={ () =>
setIntervalTaskModal(false) } interval={selectedReport}/>
    </PageWrapper>
  );
}
```

Router.ts

```
import React, { FunctionComponent } from 'react';
import {
  Switch,
  Route,
  Redirect
} from 'react-router-dom';
import { Home } from '../pages/Home';
import { Auth } from '../pages/Auth';
import { Projects } from '../pages/Projects';
import { Reports } from '../pages/Reports';
import { FormProject } from '../components/Projects/Form';
import { DisplayProject } from '../components/Projects/Display'
import { DisplayTask } from '../components/Tasks/Display'
import { BoardDisplay } from '../pages/Board'

import { useSelector } from 'react-redux';
import {
  selectToken,
} from '../store/appSlice';

export const AppRouter: FunctionComponent = () => {
  const token = useSelector(selectToken);

  const PrivateRoute = ({ component: Component, ...rest }: any) => (
    <Route
      {...rest}
      render={ props => token ? ( <Component {...props} />) :
(<Redirect to='/' />) }
    />
  );


  return (
    <Switch>
```

```
      <Route exact path="/">
        { !token ? <Auth /> : <Redirect to="/home" />}
      </Route>
      <PrivateRoute exact path="/home" component={Home}/>
      <PrivateRoute exact path="/projects" component={Projects}/>
      <PrivateRoute path="/projects/add" component={FormProject}/>
      <PrivateRoute exact path="/projects/:id"
component={DisplayProject}/>
      <PrivateRoute exact path="/reports" component={Reports}/>
      <PrivateRoute exact path="/tasks/:id" component={DisplayTask}/>
      <PrivateRoute exact path="/board/:id" component={BoardDisplay}/>
    </Switch>
  );
}
```

AppSlice.js

```
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import { API } from '../api/rest/api.config';
import { fetchUserInfoReq } from '../api/rest/queries/auth'

const token = localStorage.getItem('token');

export const fetchUser = createAsyncThunk('user/fetchUser', async () =>
{
  API.defaults.headers.common['Authorization'] = `Bearer ${token}`;
  const response = await fetchUserInfoReq();
  return response.data;
})

export const appSlice = createSlice({
  name: 'app',
  initialState: {
    token,
    userInfo: {},
    apiReady: false
  },
  reducers: {
    login: (state, action) => {
      localStorage.setItem('token', action.payload.token);
      state.token = action.payload.token;
      state.userInfo = action.payload.userInfo;
      API.defaults.headers.common['Authorization'] = `Bearer
${action.payload.token}`;
```

```
    },
    logout: state => {
      state.token = null;
      localStorage.removeItem('token');
      API.defaults.headers.common['Authorization'] = null;
    },
    updateUserInfo: (state, action) => {
      state.userInfo = action.payload;
    }
  },
  extraReducers: {
    [fetchUser.pending]: (state, action) => {
      state.status = 'loading'
    },
    [fetchUser.fulfilled]: (state, action) => {
      state.token = token;
      state.userInfo = action.payload.userInfo;
      state.apiReady = true;
    },
    [fetchUser.rejected]: (state, action) => {
      state.status = 'failed'
      state.error = action.error.message
    }
  }
})

// Action creators are generated for each case reducer function
export const { login, logout, updateUserInfo } = appSlice.actions

// The function below is called a selector and allows us to select a
value from
// the state. Selectors can also be defined inline where they're used
instead of
// in the slice file. For example: `useSelector((state) =>
state.counter.value)`
export const selectToken = state => state.app.token;
export const selectUserInfo = state => state.app.userInfo;
export const selectApiReady = state => state.app.apiReady;

export default appSlice.reducer
```

AppSliceIndex.js

```
import { configureStore } from '@reduxjs/toolkit'
```

```
import appSlice from './appSlice'

export default configureStore({
  reducer: {
    app: appSlice
  }
})
```

GlobalStyle.ts

```
import styled from 'styled-components';

export const AppWrapper = styled.section`
  height: 100%;
`;

export const LogoImage = styled.img`
  max-width: 200px;
  margin: auto;
`;

export const LogoNav = styled.img`
  height: 60px;
  margin-left: 10px;
`;

export const PageWrapper = styled.div`
  padding: 30px;
`;

export const LoadingWrapper = styled.div`
  text-align: center;
  margin-top: 30px;
`;
```

App.ts

```
import React, { FunctionComponent,useEffect } from 'react';
import { Layout } from 'antd';
import { Navigation } from './components/Navigation';
import { BrowserRouter as Router } from 'react-router-dom';
import { useDispatch } from 'react-redux';
import { fetchUser } from './store/appSlice';
const token = localStorage.getItem('token');
```

```
export const App: FunctionComponent = () => {
  const dispatch = useDispatch();

  useEffect(() => {
    if(token) {
      dispatch(fetchUser());
    }
  }, [])

  return (
    <Layout style={{ minHeight: '100vh' }}>
      <Router>
        <Navigation/>
      </Router>
    </Layout>
  );
}
```

i18n.js

```
import i18n from 'i18next';
import { initReactI18next } from 'react-i18next';
import LanguageDetector from 'i18next-browser-languagedetector';
import { pt } from './assets/lang/pt';
import { en } from './assets/lang/en';

i18n
  // detect user language
  // learn more:
https://github.com/i18next/i18next-browser-languageDetector
  .use(LanguageDetector)
  // pass the i18n instance to react-i18next.
  .use(initReactI18next)
  // init i18next
  // for all options read:
https://www.i18next.com/overview/configuration-options
  .init({
    debug: true,
    fallbackLng: 'pt',
    interpolation: {
      escapeValue: false, // not needed for react as it escapes by
default
    },
    resources: {
```

```
      en,
      pt
    }
  });

export default i18n;
```

index.ts

```
import React from 'react';
import ReactDOM from 'react-dom';
import { App } from './App';
import './index.scss';
import 'antd/dist/antd.css';
import './i18n';
import store from './store/index'
import { Provider } from 'react-redux'
import moment from 'moment';
import 'moment/locale/pt-br';
moment.locale('pt-br');

ReactDOM.render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>,
  document.getElementById("root")
);
```

eslintrc.js

```
module.exports = {
  extends: [
    'airbnb-typescript',
    'airbnb/hooks',
    'plugin:@typescript-eslint/recommended',
    'plugin:jest/recommended',
    'plugin:prettier/recommended'
  ],
  plugins: ['react', '@typescript-eslint', 'jest'],
  env: {
    browser: true,
    es6: true,
    jest: true,
```

```
  },
  globals: {
    Atomics: 'readonly',
    SharedArrayBuffer: 'readonly',
  },
  parser: '@typescript-eslint/parser',
  parserOptions: {
    ecmaFeatures: {
      jsx: true,
    },
    ecmaVersion: 2018,
    sourceType: 'module',
    project: './tsconfig.json',
  },
  rules: {
    'linebreak-style': 'off',
    'prettier/prettier': [
      'error',
      {
        endOfLine: 'auto',
      },
    ],
  },
};
```