CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE FRANCA

“Dr. THOMAZ NOVELINO”

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

VANESSA CRISTINA DIAS SILVA CAMPIONI

TORGASK:

Aplicativo para organização e gerenciamento de tarefas

Trabalho de Graduação apresentado à Faculdade de Tecnologia de Franca - “Dr. Thomaz Novelino”, como parte dos requisitos obrigatórios para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Me. Fausto Gonçalves Cintra

FRANCA/SP

2022

TORGASK:

Aplicativo para organização e gerenciamento de tarefas

Vanessa Cristina Dias Silva Campioni[1]

Resumo

Devido ao avanço exponencial da tecnologia, tanto antes quanto depois da recente pandemia pela qual passamos, as pessoas que estudam e trabalham na área foram expostas a mudanças significativas em seu estilo de vida. As pessoas têm a missão de se atualizar constantemente, aprendendo coisas novas que o mercado demanda, gerando uma quantidade enorme de tarefas profissionais em acréscimo às já existentes tarefas pessoais. Com o aumento da demanda de atividades, é necessário se organizar adequadamente para poder cumprir com tudo aquilo que é proposto na rotina e manter a produtividade no cumprimento de metas, tanto profissionais quanto pessoais. Nesse contexto, é proposto um aplicativo de gerenciamento de tarefas que visa facilitar a rotina das pessoas de tecnologia, ajudando na organização de compromissos que promovem a sua melhor execução e melhoria na produtividade. O Torgask é um aplicativo *web* que tem por objetivo organizar e gerenciar tarefas por meio de um *layout* amigável, em que o usuário pode visualizar com facilidade suas tarefas, organizá-las por nível de prioridade, vinculá-las à agenda, definir lembretes para o cumprimento das tarefas e vinculá-las a metas estabelecidas. O aplicativo oferece um contexto completo de organização e gerenciamento de tarefas para que as pessoas possam se organizar, atingir um alto nível de produtividade e entregar resultados relevantes tanto na vida profissional quanto pessoal.

Palavras-chave: Análise e Desenvolvimento de Sistemas. Desenvolvimento *Web*. Gestão de tarefas. Gestão de tempo.

Abstract

Due to the exponential advancement of technology, both before and after the recent pandemic we went through, people who study and work have been exposed to significant changes in their lifestyle. People have the mission of getting constantly up to date, learning new things, generating many professional tasks in addition to the existing personal ones. With the increase in the demand for activities, it is necessary to organize properly to be able to comply with everything that is proposed in the routine and maintain productivity in meeting goals, both professional and personal. In this context, a task management application is proposed, intending to facilitate the routine of technology people, helping in the organization of appointments that provide their better execution and improvement in productivity. Torgask is a web application that aims to organize and manage tasks through a user-friendly layout, in which the user can easily view their tasks, organize them by priority level, link them to the calendar, set reminders for the fulfillment of tasks and attaching them to established goals. The application offers a complete context of organization and task management so that people can organize themselves, achieve a high level of productivity and show relevant information for their professional and personal life.

[1] Graduanda em Análise e Desenvolvimento de Sistemas pela Fatec Dr Thomaz Novelino – Franca/SP. Endereço eletrônico: vanessa.campioni@fatec.sp.gov.br

Keywords: *Systems analysis and development. Tasks management. Time management. Web development.*

1 Introdução

O avanço da tecnologia se torna cada vez mais notório e intenso (IBGE, 2021), podemos atestar essa realidade com a recente pandemia de COVID-19, que começou em março de 2020 no Brasil e foi responsável por mudanças significativas na gestão de ensino e formas de trabalho por todo o mundo. Aderiu-se ao ensino remoto para que fosse possível a continuidade das aulas, antes feitas de modo presencial. A maior parte das empresas de tecnologia também migrou para a forma de trabalho remota.

Devido a tais adequações no processo de estudo e trabalho, surge a necessidade de adequar procedimentos que, em sua maioria, eram feitos de forma manual, visando o melhor aproveitamento do tempo e recursos disponíveis para realização de cada tarefa. Podemos tirar como exemplo a rotina de estudos que normalmente é realizada de forma manual, anotações em papel, dificuldade em encontrar um padrão para organizar os assuntos e tê-los à mão quando preciso, além da difícil tarefa de gerenciar o tempo necessário para o estudo de cada tópico e a conclusão dos objetivos definidos.

No ambiente de trabalho, mesmo para quem trabalha com tecnologia, antes não era comum o modelo remoto e, apesar de esse mercado já se encontrar em um crescimento exponencial, mesmo antes da pandemia, a nova realidade fez com que a demanda aumentasse ainda mais devido à automatização de vários tipos de procedimentos (CNN, 2021) e com tal aumento, veio a necessidade de adaptar a rotina.

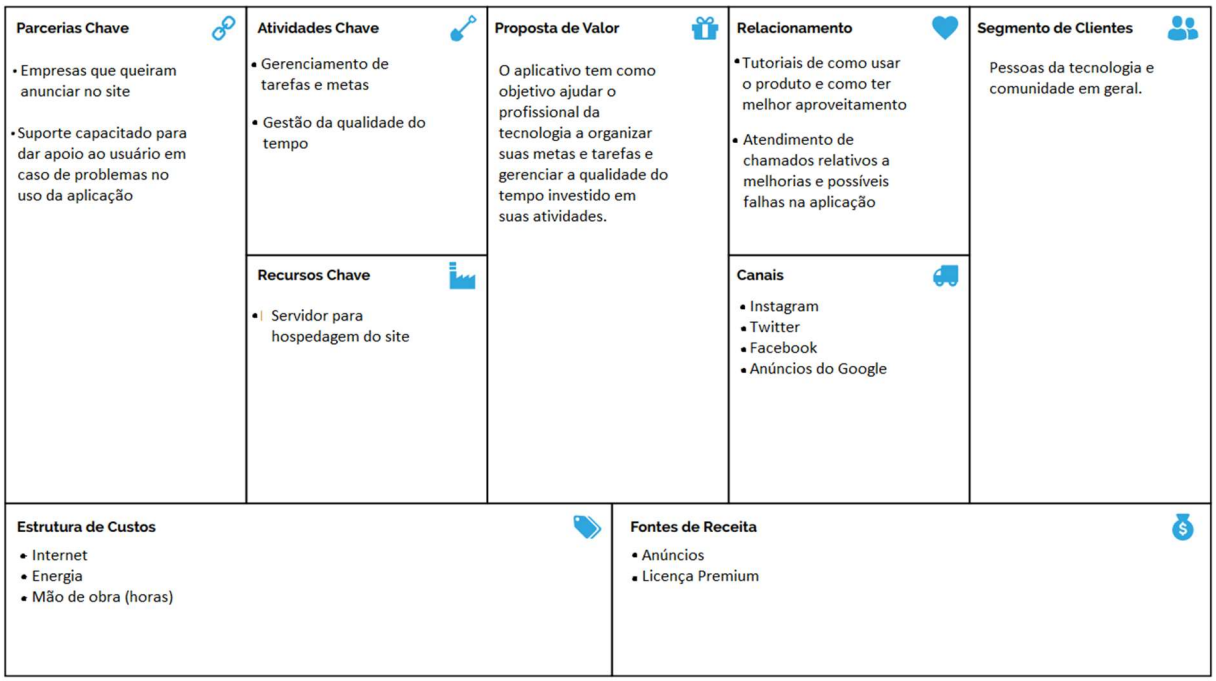
Nesse contexto, foi proposta a criação de um aplicativo *web* que tem por objetivo trazer todos os recursos necessários, de forma centralizada, para o gerenciamento do tempo de realização e execução de cada tarefa. Dessa forma, surge o Torgask, nome que vem da combinação das palavras em inglês *Task* (Tarefa) e *Organization* (Organização). A ferramenta possibilita que o usuário agende tarefas a serem realizadas, inserindo os tópicos que devem ser concluídos, anotações pertinentes e a vinculação de tarefas a metas pré-estabelecidas. Ao final, o usuário também terá em mãos uma análise de produtividade por período.

O aplicativo pretende sanar as necessidades de organização de tarefas e conteúdo de forma centralizada e prática, proporcionando ao usuário uma visão melhor do seu desempenho no cumprimento dos seus objetivos, além da praticidade de trazer, em uma ferramenta, todos os recursos necessários para o gerenciamento de suas tarefas.

2 Viabilidade do projeto

A viabilidade do projeto foi verificada através do Modelo Canvas, conforme demonstrado na Figura 1. Através deste modelo de análise, foi possível verificar com mais amplitude os pontos cruciais para que realmente fosse viável construir o gerenciador de tarefas.

Figura 1 - Modelo Canvas



Fonte: elaborado pela autora

3 Levantamento de Requisitos

3.1 Elicitação e especificação dos Requisitos

A elicitação de requisitos, de acordo com Sommerville (2007, p. 50), pode ser definida como: “O processo de derivação de requisitos de sistema através da observação de sistemas existentes, discussões com usuários potenciais e compradores, análise de tarefas etc.”

O processo de levantamento de requisitos foi desenvolvido em duas etapas. Em um primeiro momento, foi identificada a necessidade de melhoria na organização das tarefas pessoais da autora quando esta começou seus estudos para conseguir um estágio na área de tecnologia. Havia uma grande quantidade de assuntos e informações a serem assimiladas.

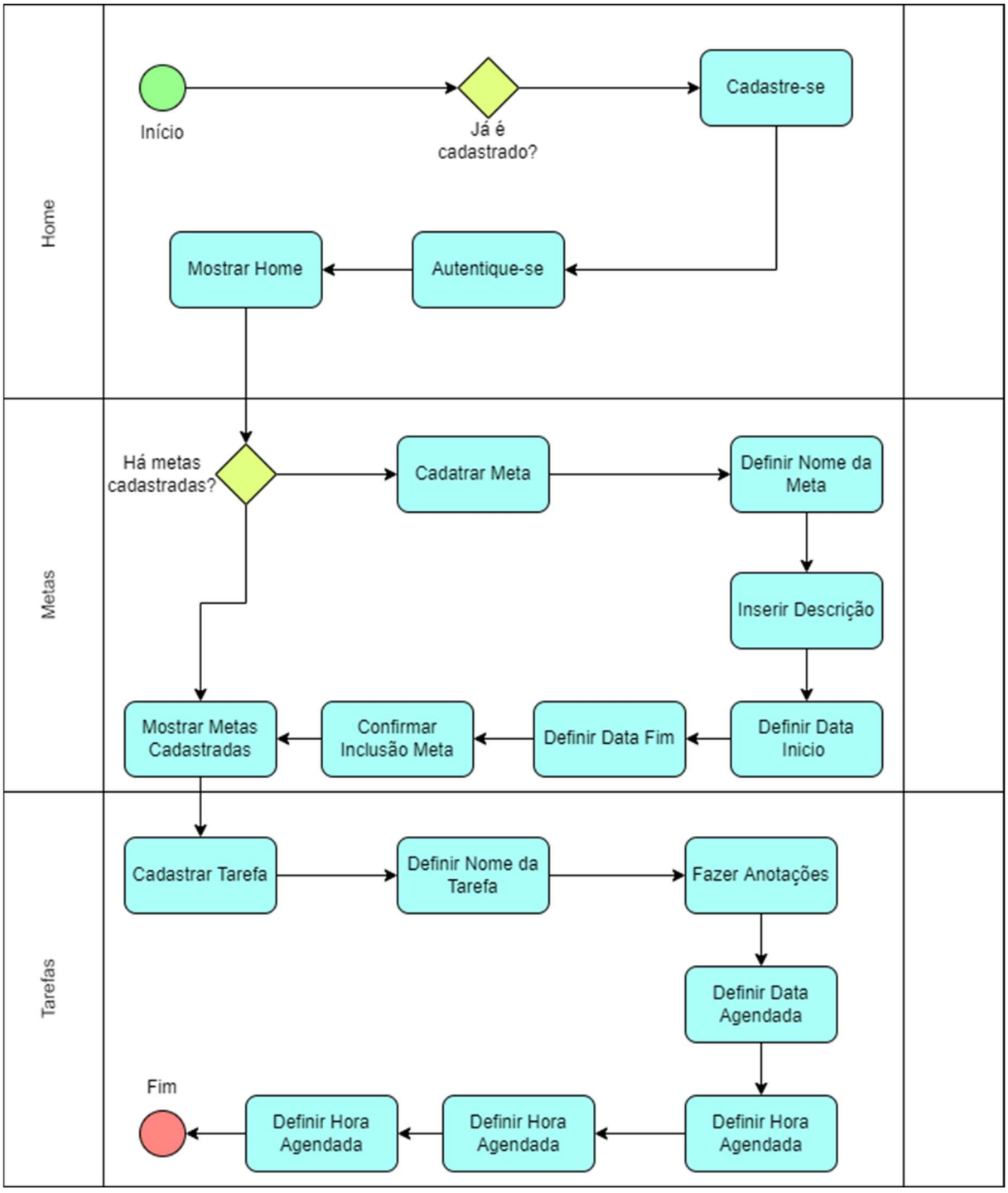
Após a identificação da necessidade, foi realizada uma reunião com três pessoas de tecnologia, tanto da área profissional quanto estudantes do campo e, nesse momento, foram levantadas as principais funcionalidades da aplicação. A ferramenta ideal deveria permitir a definição de metas de estudos e suas respectivas tarefas, assim como a inclusão dessas tarefas em um cronograma de execução, com uma visualização eficaz desse processo.

De posse de todas as informações e funcionalidades idealizadas para o sistema, foi feita a prototipação de como a aplicação seria e foram definidas a metodologia e tecnologias a serem utilizadas em seu desenvolvimento.

3.2 BPMN

Na Figura 2, temos o BPMN que demonstra o ciclo de atividades realizadas pela aplicação.

Figura 2 - BPMN



Fonte: elaborado pela autora

3.3 Requisitos Funcionais

Os tópicos de requisitos funcionais, não funcionais e as regras de negócio da aplicação, que são especificados no Quadro 1, Quadro 2 e Quadro 3, foram colhidos a partir da aplicação de entrevistas com possíveis usuários.

Quadro 1 – Requisitos Funcionais do sistema

RF001- Incluir Tarefa	Categoria: ( ) Oculto (X)Evidente	Prioridade: (X) Altíssima ( ) Alta ( ) Média ( ) Baixa
Descrição: O sistema deve permitir que o usuário cadastre uma nova tarefa através de um formulário contendo o nome, anotações, assunto, data e hora.		
RF002- Editar Tarefa	Categoria: ( ) Oculto (X) Evidente	Prioridade: (X) Altíssima ( ) Alta ( ) Média ( ) Baixa
Descrição: O sistema deve permitir que o usuário edite os dados de uma tarefa previamente cadastrada.		
RF003- Excluir Tarefa	Categoria: ( ) Oculto (X) Evidente	Prioridade: (X) Altíssima ( ) Alta ( ) Média ( ) Baixa
Descrição: O sistema deve permitir que o usuário exclua uma tarefa previamente cadastrada.		
RF004- Detalhes da tarefa	Categoria: ( ) Oculto (X) Evidente	Prioridade: ( ) Altíssima (X) Alta ( ) Média ( ) Baixa
Descrição: O sistema deve permitir que o usuário clique em algum *card* que represente a tarefa e este seja redirecionado a uma página contendo todos os detalhes da tarefa.		
RF005- Incluir Meta	Categoria: ( ) Oculto (X) Evidente	Prioridade: (X) Altíssima ( ) Alta ( ) Média ( ) Baixa
Descrição: O sistema deve permitir que o usuário inclua novas metas em seu cronograma. A meta deve conter o “nome”, “descrição” e a duração prevista.		
RF006- Editar Meta	Categoria: ( ) Oculto (X) Evidente	Prioridade: (X) Altíssima ( ) Alta ( ) Média ( ) Baixa
Descrição: O sistema deve permitir que o usuário edite os dados de uma meta previamente cadastrada.		
RF007- Excluir Meta	Categoria: ( ) Oculto (X) Evidente	Prioridade: (X) Altíssima ( ) Alta ( ) Média ( ) Baixa
Descrição: O sistema deve permitir que o usuário exclua uma meta previamente cadastrada.		
RF008- Detalhes da meta	Categoria: ( ) Oculto (X) Evidente	Prioridade: ( ) Altíssima ( ) Alta (X) Média

		( ) Baixa
Descrição: O sistema deve permitir que o usuário clique em alguma meta na lista e esse elemento se expanda permitindo que o usuário visualize todos os detalhes da meta, incluindo suas respectivas tarefas.		
RF009- Filtro Meta	Categoria: ( ) Oculto (X) Evidente	Prioridade: ( ) Altíssima ( ) Alta ( ) Média (X) Baixa
Descrição: O sistema deve dispor as metas em ordem crescente ordenadas por data de inclusão.		
RF010- Filtro Tarefa	Categoria: ( ) Oculto (X) Evidente	Prioridade: ( ) Altíssima ( ) Alta (X) Média ( ) Baixa
Descrição: O sistema deve permitir que o usuário consulte suas tarefas agendadas por período diário, semanal, mensal e anual.		
RF011- Cadastro	Categoria: ( ) Oculto (X) Evidente	Prioridade: (X) Altíssima ( ) Alta ( ) Média ( ) Baixa
Descrição: O sistema deve dispor a funcionalidade de cadastro, onde serão requisitadas informações como nome, *e-mail* e senha.		
RF012- Estado da Tarefa	Categoria: ( ) Oculto (X) Evidente	Prioridade: ( ) Altíssima ( ) Alta (X) Média ( ) Baixa
Descrição: O sistema deve permitir que o usuário marque a tarefa como concluída.		
RF013- Consulta por período	Categoria: ( ) Oculto (X) Evidente	Prioridade: ( ) Altíssima ( ) Alta (X) Média ( ) Baixa
Descrição: O sistema deve permitir que o usuário consulte suas tarefas agendadas por período diário, semanal, mensal e anual.		

3.4 Requisitos Não Funcionais

Quadro 2 – Requisitos Não Funcionais do sistema

RNF001- Plataforma Web	O sistema deve contar com uma versão *web* responsiva.	Tipo Portabilidade	( ) Desejável (X) Obrigatório	(X) Permanente ( ) Transitório
RNF002- Cores	A paleta de cores da aplicação deve ser predominantemente púrpura, cinza escuro e branco.	Tipo Interface	(X) Desejável ( ) Obrigatório	(X) Permanente ( ) Transitório
RNF003- Implementação	O sistema deverá ser desenvolvido em linguagem JavaScript, utilizando Node.js para o *back-end*, React para o *front-end* e	Tipo Implementação	( ) Desejável (x) Obrigatório	(X) Permanente ( ) Transitório

	MySQL para banco de dados.			
RNF004- Login	O aplicativo deverá autenticar o usuário através do *e-mail* e senha previamente cadastrados.	Tipo Segurança	( ) Desejável (x) Obrigatório	(X) Permanente ( ) Transitório

3.5 Regras de Negócio

Quadro 3 – Regras de Negócio do sistema.

RN001 – Assunto de estudo ou trabalho
Descrição: A pessoa de tecnologia identifica o assunto que vai ser estudado ou feito como trabalho e a partir daí define uma meta para atingir.
RN002 – Definição de etapas
Descrição: Após definida a meta, devem ser definidas as etapas, que são as tarefas necessárias para atingir àquela meta e seu respectivo tempo de execução.
RN003 – Execução
Descrição: A pessoa deve executar as tarefas propostas no prazo determinado.

3.6 Casos de Uso

Índice de casos de uso e Diagrama de casos de uso

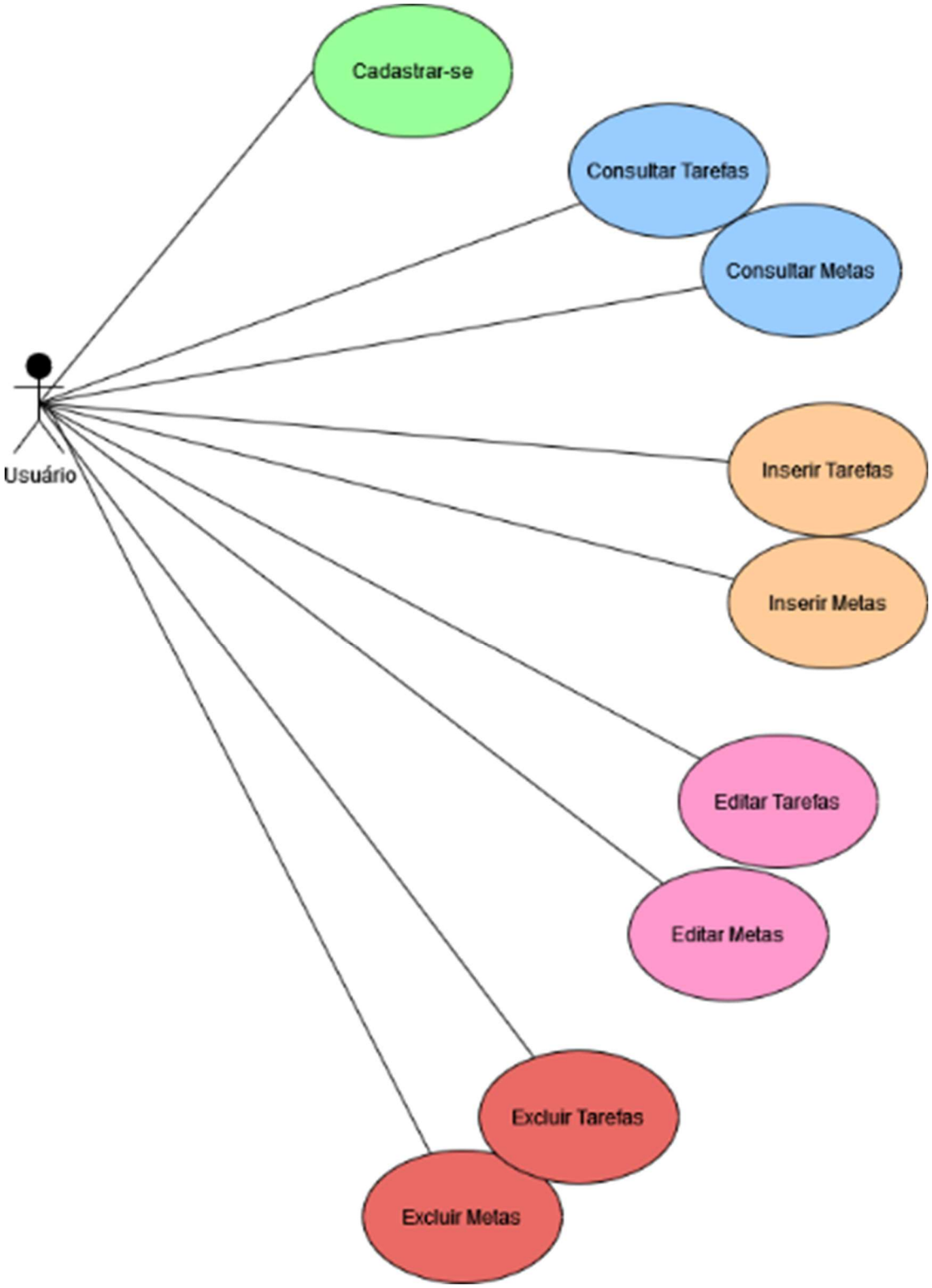
“Um caso de uso é uma sequência de interações entre o ator (alguém ou algo que interage com o sistema) e o sistema, que acontece de forma atômica, na perspectiva do ator” (DEVMEDIA, 2011).

É apresentado, na Figura 3, o diagrama de casos de uso do gerenciador de tarefas. As ações do autor consistem em um primeiro contato com o site, se cadastrar. Após o cadastro, o usuário poderá se autenticar na aplicação e começar a usar suas funcionalidades.

Partindo para as ações que podem ser executadas pelo sistema, o usuário primeiro cadastra metas, as quais podem ser editadas e excluídas posteriormente e a partir daí pode cadastrar tarefas, que também podem ser editadas e excluídas após sua criação.

Após esse processo de inserção de metas e tarefas, o usuário as tem a disposição para poder consultá-las e acompanhar seu andamento.

Figura 3 – Diagrama de Caso de Uso



Fonte: elaborado pela autora

Do Quadro 4 ao Quadro 12, temos a especificação detalhada de cada caso de uso.

Quadro 4 – Cadastrar-se

Caso de Uso – Cadastrar-se	
ID	UC 001
Descrição	Este caso de uso tem por objetivo cadastrar novos usuários
Ator Primário	Usuário do sistema
Pré-condição	Nenhuma
Cenário Principal	1. O caso de uso se inicia quando o usuário seleciona a opção cadastre-se; 2. O sistema carrega o formulário de cadastro de novo usuário; 3. O usuário informa seu nome, *e-mail* e senha; 4. O usuário confirma o cadastro; 5. O sistema armazena os dados de *login*; 6. O sistema gera um código de identificação do usuário.
Pós-condição	Nenhuma
Cenário Alternativo	1a – O usuário deixa de informar algum campo obrigatório 1a.1 O sistema identifica o campo e retorna ao usuário que o campo deve ser preenchido. 2a – O usuário informa incorretamente algum campo. 2a.1 O sistema identifica o campo e informa que deve ser preenchido de acordo com o exemplo, também apresentado junto com a mensagem de preenchimento incorreto.

Quadro 5 – Inserir Tarefas

Caso de Uso – Inserir tarefas	
ID	UC 002
Descrição	Este caso de uso tem por objetivo a inclusão de novas tarefas.
Ator Primário	Usuário do sistema
Pré-condição	Metas cadastradas
Cenário Principal	1. O *use case* inicia quando o usuário seleciona a opção incluir nova tarefa 2. O sistema carrega o formulário de inclusão de nova tarefa 3. O usuário informa o nome da tarefa, define o assunto, define a data limite para a tarefa ser concluída e faz anotações 4. O usuário confirma a inclusão da tarefa 5. O sistema armazena as informações da tarefa 6. O sistema faz a integração da tarefa com o calendário do dispositivo.
Pós-condição	Nenhuma
Cenário Alternativo	1a – O usuário não insere todas as informações obrigatórias 1a.1 O sistema não insere a nova tarefa e acusa o erro. 2a – O usuário coloca como anexos tipos de arquivos incompatíveis 2a.1 O sistema informa que o tipo de arquivo não é suportado

Quadro 6 – Inserir Metas

Caso de Uso – Inserir metas	
ID	UC 003
Descrição	Este caso de uso tem por objetivo incluir novas metas no aplicativo
Ator Primário	Usuário do sistema
Pré-condição	Usuário cadastrado no sistema
Cenário Principal	1. O *use case* inicia quando o usuário seleciona a opção inserir nova meta 2. O sistema carrega o formulário de inserção de meta 3. O usuário define o nome da meta e faz anotações descritivas. 4. O usuário define a data inicial e final para a conclusão da tarefa. 5. O usuário confirma a inclusão da meta
Pós-condição	Nenhuma
Cenário Alternativo	1a – O usuário não insere todas as informações obrigatórias 1a.1 O sistema não insere a nova tarefa e acusa o erro.

Quadro 7 – Editar Tarefas

Caso de Uso – Editar tarefas	
ID	UC 004
Descrição	Este caso de uso tem por objetivo alterar tarefas já cadastradas
Ator Primário	Usuário do sistema
Pré-condição	Ter tarefas cadastradas
Cenário Principal	1. O *use case* inicia quando o usuário clica em alguma tarefa já cadastrada e seleciona o botão alterar 2. O sistema carrega os campos da tarefa já existente e os deixa editáveis 3. O usuário realiza as alterações nos campos desejados 4. O usuário confirma a alteração 5. O sistema realiza a alteração da tarefa.
Pós-condição	Nenhuma
Cenário Alternativo	1a – O usuário deixa campos sem preenchimento 1a.1 O sistema acusa o erro que há campos obrigatórios com preenchimento incorreto 2a – O usuário não altera nenhuma informação 2a.1 O sistema inabilita o botão de salvar alteração, deixando disponível apenas o botão de sair da opção de alteração.

Quadro 8 – Editar Metas

Caso de Uso – Editar Metas	
ID	UC 005
Descrição	Este caso de uso tem por objetivo alterar metas já cadastradas
Ator Primário	Usuário do sistema
Pré-condição	Ter uma meta cadastrada
Cenário Principal	1. O *use case* inicia quando o usuário clica em alguma meta já cadastrada e seleciona o botão alterar 2. O sistema carrega os campos da meta já existente e os deixa editáveis 3. O usuário realiza as alterações nos campos desejados 4. O usuário confirma a alteração 5. O sistema realiza a alteração da meta.
Pós-condição	Nenhuma
Cenário Alternativo	O usuário deixa campos sem preenchimento 1a.1 O sistema acusa o erro que há campos obrigatórios com preenchimento incorreto 2a – O usuário não altera nenhuma informação 2a.1 O sistema inabilita o botão de salvar alteração, deixando disponível apenas o botão de sair da opção de alteração.

Quadro 9 – Excluir Meta

Caso de Uso – Excluir Meta	
ID	UC 006
Descrição	Este caso de uso tem por objetivo excluir uma meta cadastrada.
Ator Primário	Usuário do sistema
Pré-condição	Haver meta cadastrada
Cenário Principal	1. O *use case* inicia quando o usuário seleciona a opção exclusão de meta 2. O sistema pergunta ao usuário se quer mesmo excluir 3. O usuário confirma a exclusão 4. O sistema exclui a meta do banco de dados
Pós-condição	Nenhuma
Cenário Alternativo	1a. O usuário cancela a exclusão 1a.1 O sistema não exclui a meta do banco de dados.

Quadro 10 – Excluir Tarefa

Caso de Uso – Excluir Tarefa	
ID	UC 007
Descrição	Este caso de uso tem por objetivo excluir uma tarefa cadastrada.
Ator Primário	Usuário do sistema
Pré-condição	Haver tarefa cadastrada
Cenário Principal	1. O *use case* inicia quando o usuário seleciona a opção exclusão de tarefa 2. O sistema pergunta ao usuário se quer mesmo excluir 3. O usuário confirma a exclusão 4. O sistema exclui a tarefa do banco de dados
Pós-condição	Nenhuma
Cenário Alternativo	1a. O usuário cancela a exclusão

Quadro 11 – Consultar Tarefas

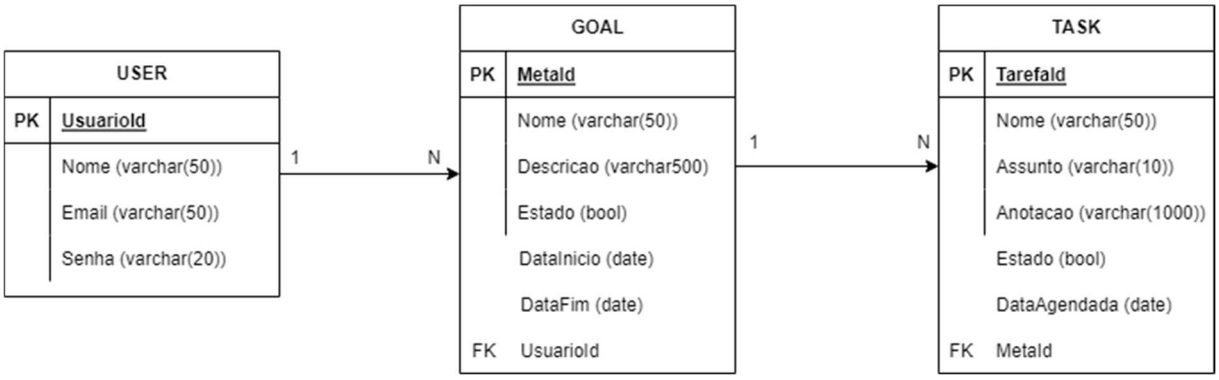
Caso de Uso – Consultar Tarefas	
ID	UC 011
Descrição	Este caso de uso tem por objetivo disponibilizar os dados das tarefas cadastradas ao usuário
Ator Primário	Usuário
Pré-condição	Ter tarefas cadastradas.
Cenário Principal	1. O *use case* inicia quando o usuário entra na página de tarefas da aplicação 2. O sistema dispõe todas as tarefas cadastradas em formato de cards de acordo com o filtro selecionado.
Pós-condição	Nenhuma
Cenário Alternativo	1a – O usuário não cadastra nenhuma meta ou tarefa; 1a.1 O sistema apresenta nenhuma tarefa. .

Quadro 12 – Consultar Metas

Caso de Uso – Armazenar dados de cadastro	
ID	UC 012
Descrição	Este caso de uso tem por objetivo disponibilizar os dados das metas cadastradas ao usuário
Ator Primário	Usuário
Pré-condição	Ter metas cadastradas
Cenário Principal	1. O *use case* inicia quando o usuário clica na aba metas na barra de navegação 2. O sistema dispõe todas as metas cadastradas em formato de lista.
Pós-condição	Nenhuma
Cenário Alternativo	1a – O usuário não cadastra nenhuma meta; 1a.1 O sistema apresenta nenhuma meta

3.7 Diagrama Entidade-Relacionamento

Figura 4 – Diagrama Entidade-Relacionamento (DER)



Fonte: elaborado pela autora

4 Ferramentas e Métodos ou Desenvolvimento

4.1 Ferramentas

a) **Adobe XD** é uma aplicação de design voltada para a usabilidade e experiência do usuário, utilizada para projetar sites e aplicativos móveis que foi criada em 2016 pela Adobe Inc (ADOBE, 2022). Foi utilizada na versão 37.1.32.2 com licença paga proveniente da Adobe. A ferramenta foi utilizada devido a prévia familiaridade e disponibilidade de conteúdo na internet para estudo de novas funcionalidades.

b) **MySQL**. Sistema de Gerenciamento de Banco de Dados criado na Suécia por David Axmark, Allan Larsson e Michael Widenius em 1980 (MYSQL, 2022). Utilizado para gerenciar os dados da aplicação na versão 8.0 com licença GPL (*General Public License*). Optou-se por utilizar essa ferramenta por se tratar de um banco de dados relacional já que a aplicação desenvolvida tem relações entre as tabelas.

c) **Node.js.** é um ambiente de execução JavaScript server-side, ou seja, com ele é possível criar aplicações que rodam sem a necessidade de um browser. Foi criado no ano de 2009 pelo pesquisador Ryan Dahl (NODEJS,

2021). Essa ferramenta foi utilizada no desenvolvimento do *back-end* da aplicação na versão 12.16.1 com licença MIT. Foi escolhida por se tratar de uma aplicação que usa a linguagem JavaScript e por ser bastante popular; sendo assim, há bastante conteúdo para pesquisa.

d) **Sequelize** é uma ORM baseada em *promises* para Node.js que suporta vários dialetos de bancos de dados (SEQUELIZE, 2021). Foi utilizada na versão 6.9.0 com licença MIT. Essa ferramenta foi escolhida devido ao seu mecanismo de migração instantâneo para criar um banco de dados e para um melhor entendimento e clareza do código devido a sua sintaxe que é bastante suscinta.

e) **Bcrypt.** É um pacote do Node.js que encripta senhas e é utilizado na segurança do acesso do usuário à aplicação, onde a senha já é salva encriptada no banco de dados (NPM, 2021). A versão utilizada foi a 5.0.1 e a justificativa de seu uso foi pensando na segurança da informação que o usuário cadastra na aplicação.

f) **JWT (*JSON Web Token*)**. É um padrão da internet que funciona como uma assinatura, em que o usuário precisa de um *token* com validade determinada para conseguir se autenticar em uma aplicação (JWT, 2021). Foi escolhido devido a sua vasta utilização atualmente no quesito de *cybersecurity*.

g) **Axios**. É um cliente HTTP baseado em promessas utilizado para conectar *back-end* com *front-end* (AXIOS, 2021). Foi utilizado na versão 0.24.0. A escolha da sua utilização foi feita pela sua base em promessas, podendo assim se aproveitar do mecanismo *async*/*await.*

h) **React.js** é uma Biblioteca JavaScript de código aberto focada na criação de interfaces em páginas web (REACT, 2021). Essa ferramenta foi utilizada para construir o *front-end* da aplicação na versão 17.0.2 com licença MIT. Foi escolhida por ser uma biblioteca que constrói telas e componentes de maneira declarativa e por manter o padrão de trabalhar com a linguagem JavaScript.

i) **Bootstrap** é um framework web com código aberto e é utilizado no desenvolvimento de componentes de interface para aplicações web

(BOOTSTRAP, 2021). Utilizado na versão 5.1.3 para estilizar alguns componentes da aplicação, sua licença é a MIT. Essa ferramenta foi escolhida por simplificar a estilização de componentes da aplicação por meio de classes contidas nessa biblioteca.

j) **StyledComponents** é uma biblioteca feita para ser utilizada no React e React Native e sua principal função é estilizar os componentes da aplicação (STYLED COMPONENTS, 2021). Utilizado na versão 5.3.3 na estilização de componentes pertencentes a aplicação, sua licença é a MIT. A ferramenta foi escolhida por ser possível definir estilos baseando-se nas propriedades de componentes React.

k) **SweetAlert** é uma API utilizada na estilização e animação de caixas de diálogos em aplicações web e mobile (SWEET ALERT, 2022). Utilizado na versão 2.1.2 para a utilização de diálogos modais. Sua licença é a MIT. A ferramenta foi escolhida por conter uma estilização muito elaborada e pela simplicidade de sua aplicação no código.

4.2 Métodos ou Desenvolvimento

4.2.1 Benchmarking

O processo de benchmarking é a avaliação do produto em relação à concorrência e foi realizado através da análise de concorrentes que já estão estabelecidos no mercado. Em primeiro lugar veio o Google Agenda, aplicativo da Google que apresenta a funcionalidade de agendar metas e tarefas em uma agenda de formato padrão. Nesse caso, foi pensado como diferencial para o Torgask, um *layout* mais dinâmico e visualmente mais agradável.

Em segundo, foi analisada a aplicação Notion que conta com a funcionalidade de realizar anotações de forma personalizada e dinâmica. Nesse caso, como o layout da aplicação já é bem enxuto, foi pensado como diferencial a facilidade no uso, já que no Notion, algumas funcionalidades apresentam um grau de complexidade bastante grande, portanto, a intenção foi trazer ao Torgask o maior grau de conteúdo intuitivo para que o usuário não tenha dificuldades ao utilizar nenhuma das funcionalidades da aplicação.

4.2.2 Branding

Branding é o processo de gestão da marca de uma empresa ou produto. O processo de branding da aplicação Torgask foi constituído por quatro etapas.

4.2.2.1 Briefing

Nessa etapa se descobre qual é a real dor do público-alvo ao qual será direcionado o produto e qual será o impacto no caso dessa necessidade não ser suprida. Além da definição do propósito do produto no mercado, essa etapa é de extrema importância para a parte comercial que virá depois, quando a aplicação já estiver em funcionamento.

Em entrevistas com estudantes e trabalhadores, foi verificado que dois pontos estavam se tornando um grande problema na hora de organizar e executar tarefas do dia a dia, sendo eles um método de organização de compromissos e a possibilidade de realizar anotações pertinentes a diversos conteúdos de forma que elas ficassem organizadas e de fácil localização.

4.2.2.2 Design Thinking

Encontrado o problema a ser resolvido, a próxima parte é pensar na melhor forma de solucioná-lo, é onde entra o Design Thinking que se trata de uma metodologia que tem como propósito encontrar as soluções ideais para os problemas dos usuários.

Para solucionar o problema encontrado, foi pensado o desenvolvimento de uma aplicação que teria como funcionalidades o agendamento de tarefas e a possibilidade de incluir diversas anotações pertinentes ao assunto daquela tarefa. Como melhor forma de construção dessa aplicação, foi definido que as tarefas seriam dispostas através de *cards* e com um sino de notificação para alertar o usuário que havia tarefas atrasadas. Sendo assim, o usuário consegue ficar alerta aos seus compromissos e consegue encontrar suas anotações com facilidade.

4.2.2.3 Logotipo

Essa parte já começa a representar a identidade da empresa e qual será a imagem que ela terá no mercado. Esse processo envolve definir a essência da

aplicação para que através de conceitos de imagens, cores e disposição de elementos, possamos passar exatamente aquilo que o produto representa para o público que for utilizá-lo.

A logotipo do Torgask foi construída de forma que os elementos transparecessem organização e sabedoria de uma maneira mais limpa e minimalista. São dois tipos de logo, a primeira, representada pela Figura 5, é a logo principal da aplicação e a segunda, representada pela Figura 6, é a logo alternativa. A escolha de uso de cada uma foi feita de acordo com o fundo em que seriam colocadas. Para fundos mais escuros utilizou-se a logo principal e para fundos mais claros, a alternativa.

Figura 5 – Logo Principal

Figura 6 – Logo Alternativa

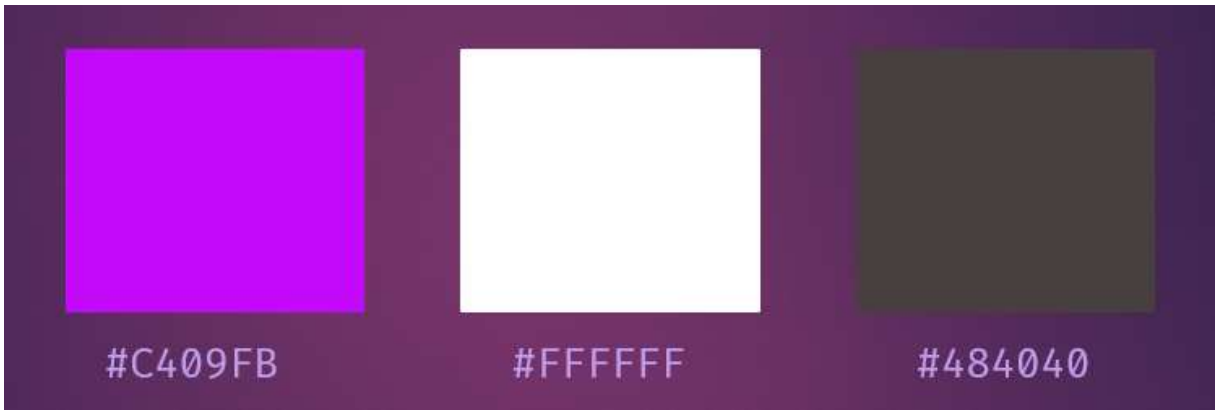
4.2.2.4 Interface

A partir da definição dos tópicos anteriores, já conseguimos definir com mais facilidade qual será a aparência da aplicação, nos atentando aos pontos de diferenciais, melhores práticas para solucionar o problema do usuário e as melhores cores e padrões para comunicar exatamente a essência do que é o produto.

Para construir o visual da aplicação, foram considerados dois pontos muito importantes que são a disposição dos elementos da tela, de forma que fique intuitivo para o usuário utilizar as funcionalidades e a paleta de cores para que fosse possível transparecer a imagem de organização e sabedoria que o produto almejava levar para a vida dos usuários.

Foi definido que a melhor forma de disposição dos elementos seria através de *cards,* conforme demonstrado na Figura 26 que contém a tela de tarefas da aplicação. E, sobre a paleta de cores, demonstrada na Figura 7, foram escolhidas as cores purpura (#C409FB) remetendo a criatividade e sabedoria, cinza (#484040) trazendo neutralidade para intensificar as demais cores e o branco (#FFFFFF) para quebrar os tons mais escuros e harmonizar melhor a interface (VENTURUS, 2020).

Figura 7 – Paleta de Cores



4.2.3 Desenvolvimento

A linguagem de programação utilizada para o desenvolvimento da aplicação web foi JavaScript. No *back-end* foi desenvolvida uma API em Node.js para receber as requisições do cliente através do navegador e retornar os resultados esperados. Os dados gerados são armazenados no banco de dados MySQL. Já para o *front-end* foi utilizada a biblioteca React para criar as telas e componentes da aplicação.

4.2.3.1 *Back-end*

Para a criação do banco de dados, foram utilizadas as *migrations*, a partir das quais o banco é criado a partir do próprio código. Um exemplo da criação da tabela de *users* por *migrations* é demonstrado na Figura 8.

Figura 8 – Demonstração de *migrations*

```
module.exports = {
  up: async (queryInterface, Sequelize) => {
    await queryInterface.createTable('users', {
     id: {
       type: Sequelize.INTEGER,
       autoIncrement: true,
       allowNull: false,
       primaryKey: true
     },
     nome: {
       type:Sequelize.STRING,
       allowNull:false
     },
     email: {
       type: Sequelize.STRING,
       allowNull:false
     },
     senha: {
       type: Sequelize.STRING,
       allowNull: false
     },
     created_at: {
       type: Sequelize.DATE,
       allowNull: false
     },
     updated_at: {
      type: Sequelize.DATE,
      allowNull: false
     }

     });

  },

  down: async (queryInterface, Sequelize) => {

    await queryInterface.dropTable('users');
```

Fonte: elaborado pela autora

Após a criação do banco, partimos para a construção do processo onde o usuário pode se cadastrar e posteriormente se autenticar para poder utilizar todas as funcionalidades do gerenciador de tarefas. Na Figura 9 é demonstrado o processo de autenticação que a aplicação percorre ao ter um usuário cadastrado:

Figura 9 – Função de encriptação

```
const User = require('../models/User');
const bcrypt = require('bcrypt');
const saltRounds = 10;
const jwt = require('jsonwebtoken');
const jwtConfig = require('../config/jwt.json');

const hashPassword = function (plainPassword) {
    return bcrypt.hashSync(plainPassword, saltRounds);
};

const decryptPassword = function (plainPassword, hashFromDB) {
    return bcrypt.compareSync(plainPassword, hashFromDB);
}
```

Fonte: elaborado pela autora

Nesta Figura 10, são declaradas as funções responsáveis para encriptar (*hashPassword*) e decriptar a senha que o usuário informará em seu cadastro. Também é implementado o *token* JWT para trazer segurança na autenticação do usuário quando faz seu *login* na aplicação.

Figura 10 – Implementação do *token* com JWT

```
const user = await User.findOne({ where: { email: email } });
if (user) {
    if (decryptPassword(senha, user.senha)) {

        //JWT
        const token = jwt.sign({ id: User.id }, jwtConfig.secret, {
           expiresIn: 1200,
        });
        return res.status(200).json({
            status: 200,
            token,
            user
        });
    }
    return res.status(401).json({
        status: 401,
        message: "Senha Inválida!"
    });
}
return res.status(404).json({
    status: 404,
    message: "E-mail não encontrado!"
});
}
catch(err) {
    return commonErrorResponse(err, res);
}
```

Fonte: elaborado pela autora

Além da tabela de *users*, a aplicação é constituída por mais duas tabelas que se relacionam entre si, sendo elas a tabela de *Goals* representando as metas e a tabela de *Tasks* representando as tarefas. Para realizar as associações, foi utilizada a sintaxe do Sequelize nas *models* (abstração das entidades feitas para o código) das tabelas criadas, como demonstram as Figuras 11 e 12.

Figura 11 – Criação da tabela Tasks

```
class Task extends Model {
    static init(sequelize) {
        super.init({
            nome: DataTypes.STRING,
            anotacao: DataTypes.STRING,
            estado: DataTypes.BOOLEAN,
            data_agendada: DataTypes.DATE,
            goal_id: DataTypes.INTEGER
        }, {
            sequelize
        })
    }

    static associate(models) {
        this.belongsTo(models.Goal, { foreignKey: 'goal_id', as: 'goal' })
    }
}
```

Fonte: elaborado pela autora

Figura 12 – Criação da tabela Goals

```
class Goal extends Model {
    static init(sequelize) {
        super.init({
            nome: DataTypes.STRING,
            descricao: DataTypes.STRING,
            estado: DataTypes.BOOLEAN,
            data_inicio: DataTypes.DATE,
            data_fim: DataTypes.DATE,
        }, {
            sequelize
        })
    }

    static associate(models) {
        this.belongsTo(models.User, { foreignKey: 'user_id', as: 'user' }),
        this.hasMany(models.Task, { foreignKey: 'goal_id', as: 'tasks' } )
    }
}
```

Fonte: elaborado pela autora

Após as *models*, foi a hora de criar os *controllers*, que são responsáveis por intermediar a relação entre *views* (telas) e *models*, recebendo e tratando requisições, com o que a *model* retorna e, posteriormente, devolve o resultado para o usuário. Na Figura 13 é demonstrada uma função do *controller* de Tasks que é responsável por criar uma tarefa.

Figura 13 – Função para criar tarefas no *controller*

```
async create(req, res) {
    const { goal_id } = req.params;

    const { nome, anotacao, estado, data_agendada } = req.body;

    const goal = await Goal.findByPk(goal_id);
    if (!goal) {
        return res.status(400).json({ error: 'Cadastre uma meta primeiro!' });
    }

    const task = await Task.create({
        nome,
        anotacao,
        estado,
        data_agendada,
        goal_id,
    });

    return res.json(task);
},
```

Fonte: elaborado pela autora

Por fim, foi a hora de definir as rotas, que são responsáveis por possibilitar o acesso aos resultados que a API gera em retorno as requisições realizadas, como demonstrado na Figura 14.

Figura 14 – Rotas da API

```
// Login
routes.post('/authenticate', UserController.login);

// Users
routes.get('/users', auth, UserController.index);
routes.post('/users', UserController.create);

// Goals
routes.get('/goals', auth, GoalsController.index);
routes.post('/goals', auth, GoalsController.create);
routes.put('/goals/:id', auth, GoalsController.update);
routes.delete('/goals/:id', auth, GoalsController.delete);
routes.get('/goals/:id/getByGoal', auth, TaskController.getByGoal);

// Tasks
routes.get('/tasks/:id', auth, TaskController.getById);
routes.post('/tasks', auth, TaskController.create);
routes.put('/tasks/:id', auth, TaskController.update);
routes.delete('/tasks/:id', auth, TaskController.delete);
routes.get('/tasks/filter/late', auth, TaskController.late);
routes.get('/tasks/filter/index', auth, TaskController.index);
routes.get('/tasks/filter/today', auth, TaskController.today);
routes.get('/tasks/filter/week', auth, TaskController.week);
routes.get('/tasks/filter/month', auth, TaskController.month);
routes.get('/tasks/filter/year', auth, TaskController.year);


module.exports = routes;
```

Fonte: elaborado pela autora

A função *auth*, demonstrada na Figura 15, está instanciando o JWT, sendo responsável por só permitir que o usuário acesse as rotas da API se estiver autenticado com um *token* válido.

Figura 15 – Function *auth*

```
function auth(req, res, next) {
    const token = req.headers['x-access-token'];
    jwt.verify(token, jwtConfig.secret, (err, decoded) => {
        if(err) return res.status(401).send({ error: 'Token inválido'});

        req.user_id = decoded.user_id;
        next();
    });
}
```

Fonte: elaborado pela autora

4.2.3.2 *Front-end*

O *front-end*, que é a parte visual da aplicação, foi desenvolvida utilizando React.js. O primeiro passo foi a criação dos componentes da aplicação, que são os elementos HTML que vão ser utilizados com frequência nas *views*.

A Figura 16 demonstra a criação do componente *Header*, que vai no topo de todas as *views*:

Figura 16 – Demonstração da criação de componentes

```
function Header({ lateCount, clickNotification }) {
    return (
      <S.Container>
        <S.LeftSide>
          <img src={logo} alt="Logo" />
        </S.LeftSide>

        <S.RightSide>
          <Link to="/home">INÍCIO</Link>
          <span className="dividir" />
          <Link to="/goals">METAS</Link>
          <span className="dividir" />
          <Link to="/tasks">TAREFAS</Link>
          <span className="dividir" />
          <Link to="/tasks/filter/late">
            <button id="notification">
              <img src={bell} alt="Notificacao" />
              <span>{lateCount}</span>
            </button>
          </Link>
        </S.RightSide>

      </S.Container>
    )
}
```

Fonte: elaborado pela autora

E uma parte da estilização utilizando a ferramenta *StyledComponents*, como demonstrado na Figura 17. Pode-se notar que os componentes React são tratados diretamente na estilização.

Figura 17 – Demonstração da estilização com *StyledComponents*

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

export const Container = styled.div`
    width: 100%;
    height: 70px;
    background: #C409FB;
    border-bottom: 5px solid #FFFFFF;

    display: flex;
`

export const LeftSide = styled.div`
    width: 50%;
    height: 70px;
    display: flex;
    align-items: center;
    padding-left: 5px;

    img {
        width: 170px;
        height: auto;
    }
`
```

Fonte: elaborado pela autora

Após a criação dos componentes, foi a vez de criar as *views* do projeto, que são as telas. Nelas são importados os componentes criados anteriormente, como segue a demonstração na Figura 18.

Figura 18 – Importação de componentes nas *views*

```
return (

  <S.Container>

    <Header lateCount={lateCount} clickNotification={Notification} />


    <S.Title>
      <h3>DETALHES DA TAREFA</h3>
    </S.Title>

    <S.DetailsArea>
      <S.DetailsBox>
        <div className='nome-tarefa'>
          <h3>Nome</h3>
        </div>
        <div className='anotacao-tarefa'>
          <div className='texto-anotacao'>Texto</div>
        </div>
```

Fonte: elaborado pela autora

Para finalizar, apenas foi necessário definir as rotas de acesso às *views,* conforme demonstrado na Figura 19, e importar estas no arquivo *index*, que foi o responsável por renderizar os componentes e telas da aplicação.

Figura 19 – Rotas da aplicação

```
export default function Routes() {

    return(
        <BrowserRouter>
            <Switch>
                <Route path="/authenticate" exact component={Login} />
                <Route path="/register" exact component={Register} />
                <Route path="/home" exact component={Home} />
                <Route path="/tasks" exact component={Task} />
                <Route path="/goal" exact component={NewGoal} />
                <Route path="/task" exact component={NewTask} />
                <Route path="/tasks/:id" exact component={TaskDetails} />
                <Route path="/goals" exact component={Goal} />
                <Route path="/goals/:id" exact component={GoalDetails} />
                <Route path="/tasks/filter/late" exact component={LateTask
                <Route path="/error" exact component={NotFound} />

            </Switch>
        </BrowserRouter>
    )
}
```

Fonte: elaborado pela autora

Além da construção de componentes, telas e o acesso a essas telas, foi criado na pasta *services* o arquivo api.js que é responsável por se conectar com o *back-end*, utilizando o pacote Axios, e assim ser possível a automação das telas criadas, aplicando a elas funcionalidades provenientes da API, de acordo com a Figura 20.

Figura 20 – Conexão com a API utilizando Axios

```
import axios from 'axios';

const api = axios.create({
    baseURL: 'http://localhost:3333'
});

export default api;
```

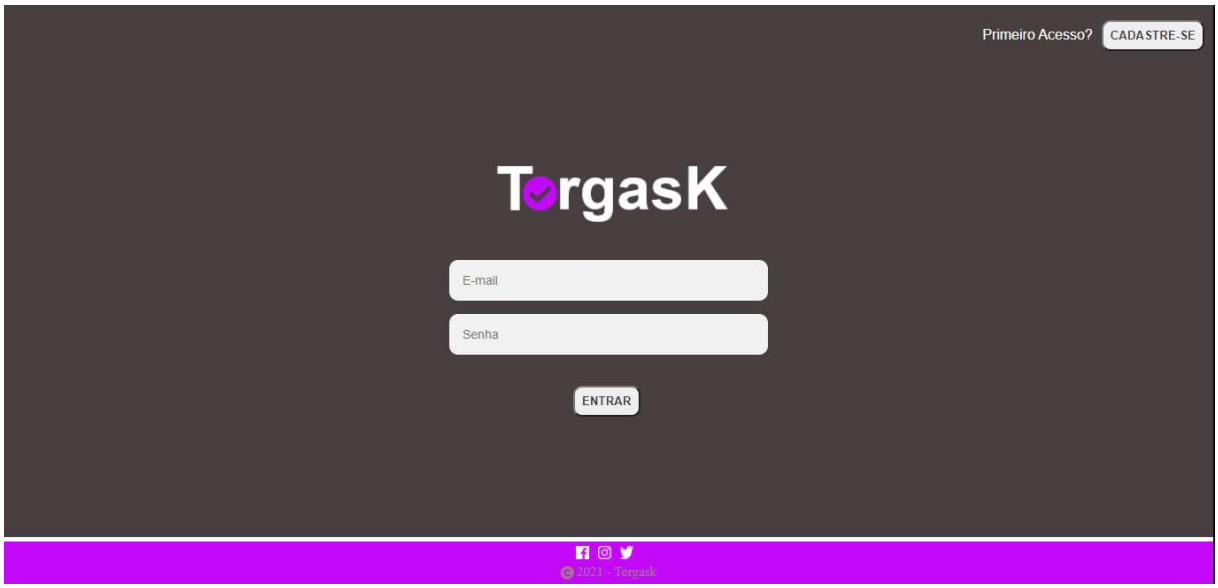
Fonte: elaborado pela autora

Os testes foram feitos durante e após a construção da aplicação, sendo estes voltados para o correto funcionamento das funcionalidades, a análise da usabilidade e da segurança.

5 Resultados e Discussão

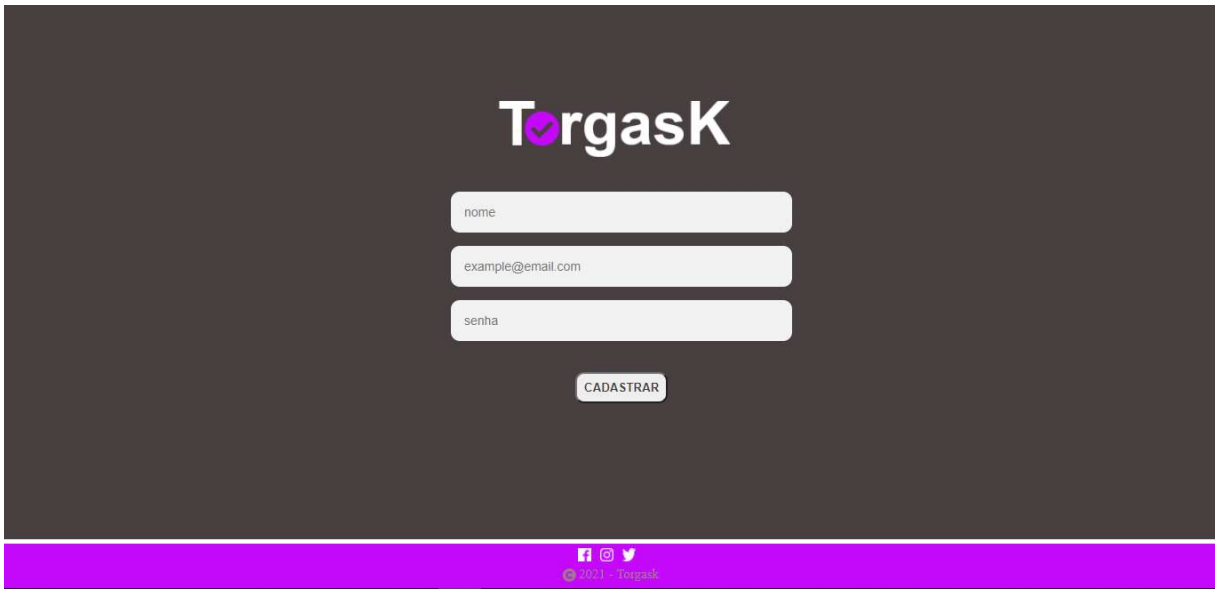
Iniciamos a aplicação com a tela de autenticação, demonstrada pela Figura 21, onde o usuário pode entrar no sistema, caso tenha um cadastro, ou então clicar no botão cadastre-se, sendo redirecionado para a tela de cadastro, representada na Figura 22, onde pode realizar o seu cadastro de usuário e começar a utilizar as funcionalidades do sistema.

Figura 21 – Tela de *Login*



Fonte: elaborado pela autora

Figura 22 – Tela de Cadastro de Usuário



Fonte: elaborado pela autora

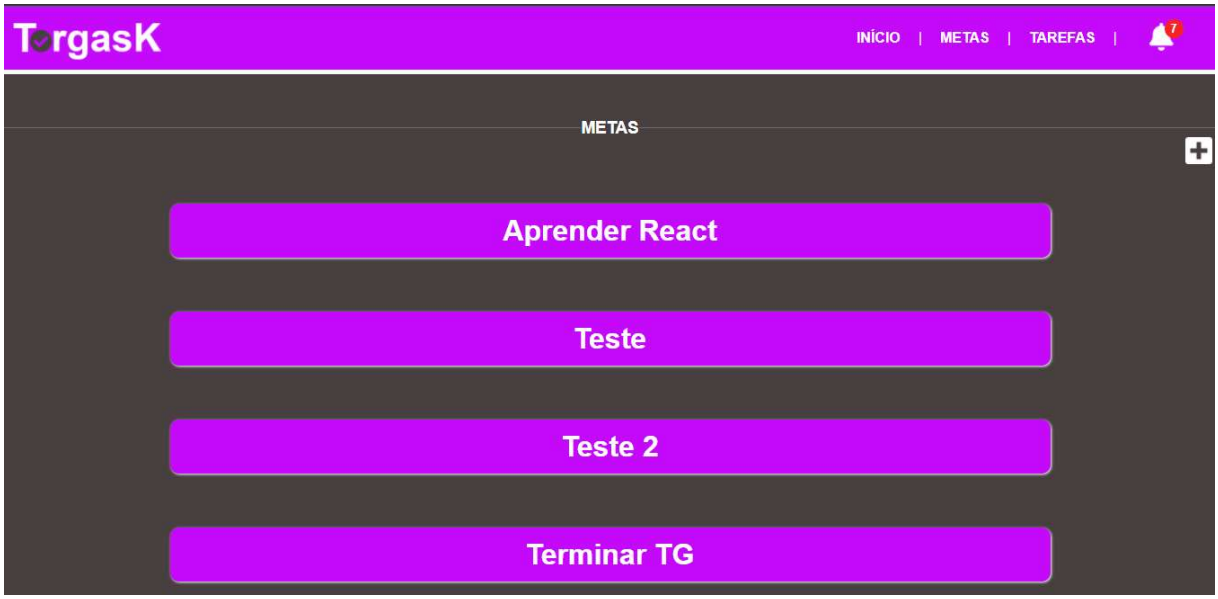
Após se autenticar, o usuário é redirecionado para a tela *Home*, representada pela Figura 23, onde há disponível um tutorial de primeiros passos em formato de vídeo. Para acessá-lo, basta clicar no botão PRIMEIROS PASSOS e já será aberto o vídeo instrutivo na plataforma do YouTube. Caso o usuário já saiba como utilizar a aplicação, são disponibilizadas na barra de navegação, localizada no canto superior direito da tela, as opções de metas, tarefas e o sino de notificações que representa as tarefas cadastradas que já estão atrasadas.

Figura 23 – Tela *Home*

Fonte: elaborado pela autora

Selecionando a opção Metas, o usuário tem acesso a todas as metas cadastradas, conforme demonstrado a Figura 24 e pode clicar em algum *card* de meta para ver seus detalhes, editá-las ou exclui-las, como na Figura 25, ou então clicar no ícone com o sinal de + no canto superior direito para cadastrar uma nova meta, conforme a Figura 26.

Figura 24 – Tela de Metas Cadastradas

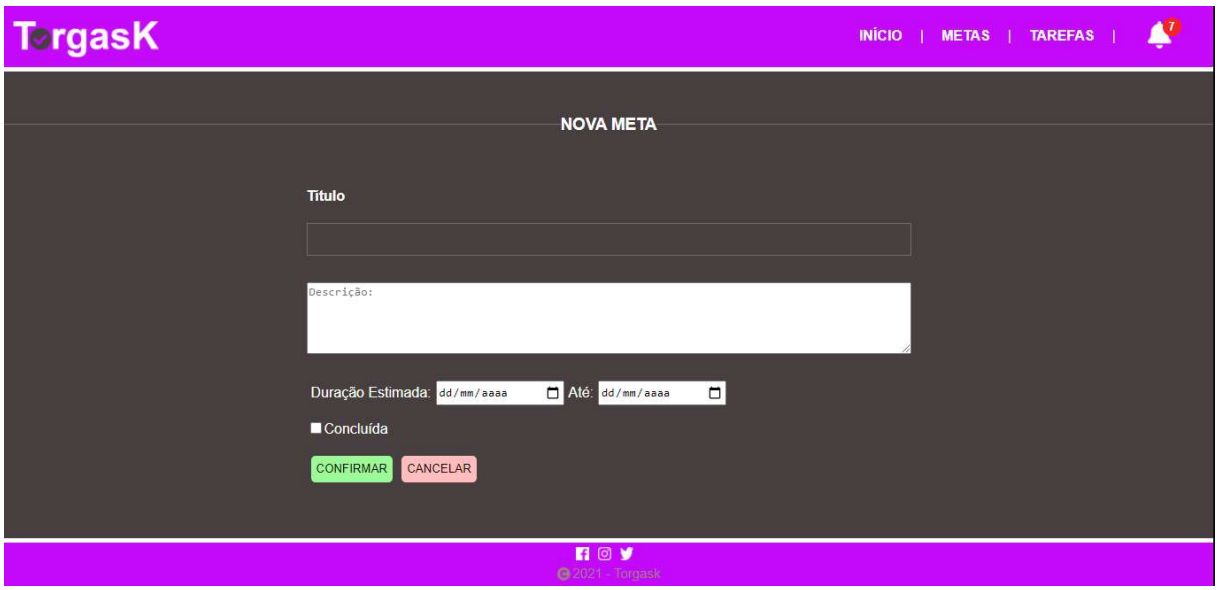


Fonte: elaborado pela autora

Figura 25 – Tela de Detalhes da Meta

Fonte: elaborado pela autora

Figura 26 – Tela de Cadastro Nova Meta

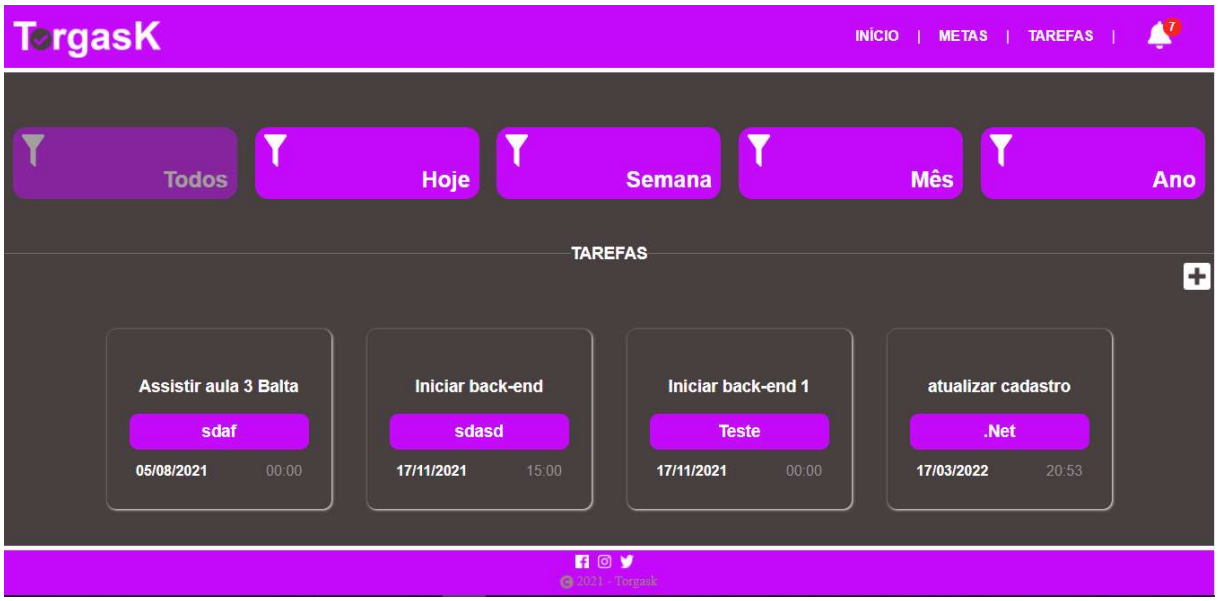


Fonte: elaborado pela autora

Já quando a opção Tarefas for selecionada, o usuário será redirecionado para a tela que apresenta todas as tarefas cadastradas através de *cards* (Figura 24). O usuário poderá filtrar as tarefas por dia, semana, mês, ano, para isso, basta que o usuário clique em um dos filtros localizados na parte superior da tela, logo abaixo da barra de navegação. Além de visualizar todas as tarefas criadas, pode-se criar outras clicando no botão + no canto superior direito.

Ao clicar em um *card* de tarefas, o usuário pode visualizar seus detalhes, editá-las ou exclui-las, como demonstrado na Figura 27.

Figura 27 – Tela de Tarefas



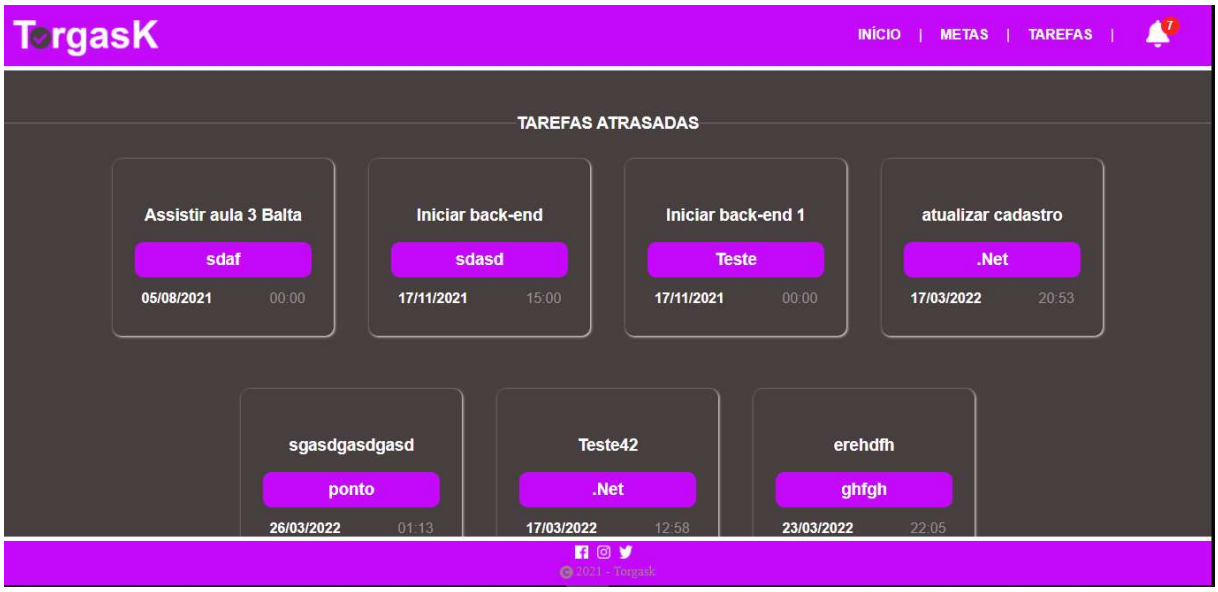
Fonte: elaborado pela autora

Figura 28 – Tela Detalhes de Tarefa

Fonte: elaborado pela autora.

Por último, ao clicar no sino de notificação, são mostradas as tarefas que estão em atraso, conforme a Figura 29.

Figura 29 – Tela de Tarefas Atrasadas



Fonte: elaborado pela autora

Considerações finais

A grande demanda por automatização de processos trouxe a necessidade de constante atualização e movimento por parte de estudantes e profissionais. No entanto, a tarefa de conciliar as vidas pessoal e profissional pode se tornar um grande fardo, caso não haja organização nesse fluxo de trabalho, estudos e lazer. É nesse contexto que o Torgask mostra sua viabilidade. Um gerenciador de tarefas que organiza as demandas do dia a dia de forma intuitiva e prática, possibilitando a visualização de todas as demandas, o gerenciamento do tempo gasto para realizá-las e ainda auxiliando no planejamento futuro com a possibilidade de inclusão de metas.

Essa aplicação facilita a vida do profissional de tecnologia ao tirar a responsabilidade de contabilizar a quantidade de tarefas cotidianas, o tempo que será gasto para realizá-las, a organização de cada tarefa em relação ao todo e a forma como o conteúdo de cada tarefa será monitorado e armazenado, possibilitando uma melhor gestão de tempo e mantendo a produtividade.

O próximo passo é implementar uma versão *mobile* do Torgask para que a aplicação fique ainda mais utilizável, estando disponível na palma da mão do usuário a qualquer momento do dia.

Referências

ADOBE. Adobe XD. Disponível em: https://www.adobe.com/br/products/xd.html. Acesso em 27 mar. 2022.

AXIOS. **Axios API**. Disponível em: https://axios-http.com/ptbr/docs/api_intro. Acesso em: 21 nov. 2021.

BOOTSTRAP. **Bootstrap**. Disponível em: https://getbootstrap.com/. Acesso em: 20 dez. 2021.

CNN, **Procura por profissionais de tecnologia cresce 671% durante a pandemia**, 27 out. 2021. Disponível em: https://www.cnnbrasil.com.br/business/procura-por-profissionais-de-tecnologia-cresce-671-durante-a-pandemia/. Acesso em: 27 mar. 2022.

DEVMEDIA, **Especificação de Casos de Uso - Engenharia de Software 32**, 2011. Disponível em: https://www.devmedia.com.br/especificacao-de-casos-de-uso-engenharia-de-software-32/19012. Acesso em: 22 jun. 2022.

IBGE [INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATISTICA]. **Pesquisa Nacional por Amostra de Domicílios (PNAD),** 14 abr. 2021**.** Disponível em: https://www.gov.br/mcom/pt-br/noticias/2021/abril/pesquisa-mostra-que-82-7-dos-domicilios-brasileiros-tem-acesso-a-internet. Acesso em: 7 ago. 2021.

JWT. **Introduction to JSON Web Tokens**. Disponível em: https://jwt.io/introduction. Acesso em: 20 out. 2021.

MYSQL. **MySQL**. Disponível em: https://www.mysql.com/. Acesso em 27 mar. 2022.

NODEJS, **Node.js**. Disponível em: https://nodejs.org/. Acesso em: 20 out. 2021.

NPM. **Bcrypt**. Disponível em: https://www.npmjs.com/package/bcrypt. Acesso em: 20 out. 2021.

REACT. **React**: uma biblioteca JavaScript para criar interfaces de usuário. Disponível em: https://pt-br.reactjs.org/. Acesso em: 20 dez. 2021.

SEQUELIZE. **Sequelize**. Disponível em: https://sequelize.org/. Acesso em: 20 out. 2021.

SOMMERVILLE, Ian. **Engenharia de Software**. São Paulo: Pearson Prentice Hall, 2011.

STYLED COMPONENTS. **Styled Components**. Disponível em: https://styled-components.com/. Acesso em: 20 dez. 2021.

SWEET ALERT. **Sweet Alert**. Disponível em: https://sweetalert.js.org/. Acesso em: 15 mar. 2022.

VENTURUS. **Psicologia das Cores**. Disponível em: https://venturus.org.br/psicologia-das-cores/. Acesso em: 23 jun. 2022.