GHABRIELL LUZ ALVES DA SILVA

DESENVOLVIMENTO DE UM SISTEMA DE SERVICE DESK NA EMPRESA SYDLE

LAVRAS – MG

2023

GHABRIELL LUZ ALVES DA SILVA

DESENVOLVIMENTO DE UM SISTEMA DE SERVICE DESK NA EMPRESA SYDLE

Relatório de estágio supervisionado apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para obtenção do título de Bacharel.

Prof. DSc. Joaquim Quinteiro Uchôa
Orientador

LAVRAS – MG

2023

GHABRIELL LUZ ALVES DA SILVA

DESENVOLVIMENTO DE UM SISTEMA DE SERVICE DESK NA EMPRESA SYDLE

Relatório de estágio supervisionado apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para obtenção do título de Bacharel.

APROVADA em 27 de Fevereiro de 2023.

Prof. DSc. Joaquim Quinteiro Uchôa	DAC
Prof. DSc. Paulo Afonso Parreira Junior	DCC
Profa. DSc. Renata Teles Moreira	DCC

gov.br
Documento assinado digitalmente
JOAQUIM QUINTEIRO UCHOA
Data: 15/03/2023 19:25:43-0300
Verifique em https://validar.iti.gov.br

Prof. DSc. Joaquim Quinteiro Uchôa
Orientador

LAVRAS – MG
2023

AGRADECIMENTOS

Agradeço aos meus pais, pelo apoio e amor incondicional.
Aos meus professores, por compartilharem comigo seu bem mais precioso, o conhecimento.
Ao meu orientador, professor Joaquim, pelo tempo dedicado em me ajudar na realização do meu maior sonho.
À minha namorada, Ingrid, pelo apoio e incentivo em todos os momentos.
Aos meus pets, Ícaro e Gaia, pelo apoio emocional e por estarem comigo sempre que precisei.
Por fim, agradeço à empresa SYDLE pela oportunidade de estágio e aos meus colegas de equipe pela troca de conhecimento.

Quando você quer alguma coisa, todo o universo conspira para que você realize seu desejo.
Paulo Coelho

RESUMO

Este relatório de estágio supervisionado busca descrever as atividades realizadas na empresa SYDLE, localizada em Belo Horizonte - MG. A empresa oferece uma plataforma de *service desk*, que centraliza o relacionamento entre uma empresa e seus clientes/funcionários, atuando como meio de contato e automatizando processos. Durante o estágio, foi realizada a implementação de novas funcionalidades no sistema, utilizando o framework front-end Stencil e a linguagem TypeScript. Além disso, foi utilizado o SYDLE ONE, plataforma criada pela empresa que simplifica o back-end e gerenciamento de banco de dados, com recursos como a criação de classes e objetos, fornecimento de API REST para o acesso aos mesmos, entre outras.

Palavras-chave: Desenvolvimento Web. Stencil. TypeScript. Service desk. SYDLE ONE.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

O seguinte trabalho descreve os aprendizados e experiências vivenciados pelo aluno durante seu período de estágio não-obrigatório como engenheiro de *software* na empresa SYDLE.

A SYDLE é uma empresa de tecnologia fundada em 2005, sediada em Belo Horizonte, no estado de Minas Gerais. A empresa trabalha com desenvolvimento de *software*, oferecendo soluções em automação e gestão de processos, ferramentas de análise de dados, *e-commerce*, *service desk*, entre outros. Além disso, possui a plataforma SYDLE ONE, que integra todas essas soluções em um único lugar. Possui clientes em mais de 80 países, entre eles: Bayer, Carrefour e Petrobras.

O estágio foi realizado de maneira remota, em *home office*. Como forma de comunicação com os membros da empresa, o aluno utilizou sistemas de *chat* e *e-mail*, além de chamadas de áudio e vídeo.

Durante o estágio, o aluno atuou no desenvolvimento de um *service desk*, sistema que funciona como canal de atendimento, centralizando todas as demandas de uma empresa em um único lugar. Nesse processo, utilizou tecnologias que desconhecia ou possuía pouca familiaridade até então, adquirindo conhecimento em diversas áreas.

O documento está organizado como descrito a seguir: No Capítulo 2, são apresentadas os conhecimentos técnicos necessários ao decorrer do estágio, familiarizando o leitor com os conceitos apresentados durante o trabalho. No Capítulo 3, são descritas as atividades desenvolvidas pelo aluno como estagiário. No Capítulo 4, é feita uma análise, buscando pontuar quais conhecimentos prévios auxiliaram o aluno no processo e qual o impacto da experiência para sua formação acadêmica e profissional.

2 REFERENCIAL TEÓRICO

Neste capítulo, são apresentados e detalhados as metodologias e tecnologias utilizadas durante o período de estágio, necessários para o pleno entendimento deste relatório.

2.1 Metodologias e conceitos

Nesta seção, são apresentadas metodologias e conceitos utilizados durante o desenvolvimento do projeto.

2.1.1 Scrum

Segundo Schwaber e Sutherland (2020), o Scrum é um *framework* leve que ajuda pessoas, times e organizações a gerarem valor através de soluções adaptativas para problemas complexos. Amplamente utilizado na indústria do desenvolvimento de *software*, este framework possui como principais pontos o *Scrum Team*, os *Scrum Events* e os *Scrum Artifacts*.

2.1.1.1 Scrum Team

O *Scrum Team* é um pequeno time autogerenciável de pessoas focadas no desenvolvimento do produto, onde cada membro possui uma responsabilidade específica, sendo elas o *Scrum Master*, o *Product Owner* e os Desenvolvedores.

- *Scrum Master*: É responsável por garantir o funcionamento do Scrum, auxiliando os demais membros a compreender sua teoria, removendo impedimentos e promovendo os *Scrum Events* de maneira proveitosa.
- *Product Owner*: É responsável por maximizar o valor gerado e gerenciar o *Product Backlog*, assumindo tarefas como criar itens, definir suas prioridades e comunicá-los ao *Scrum Team*.
- Desenvolvedores: São responsáveis pelo desenvolvimento das funcionalidades do produto, onde a cada nova *sprint* definem quais serão as tarefas executadas.

2.1.1.2 Scrum Events

Os *Scrum Events* são cerimônias que ocorrem de forma periódica, onde cada uma possui um objetivo dentro do Scrum:

- *Sprint*: São ciclos de desenvolvimento que costumam durar de duas a quatro semanas, onde toda a equipe trabalha visando entregar um incremento funcional do produto.
- *Sprint Planning*: Definirá as atividades que serão desenvolvidas durante a *sprint*. Nela são discutidos os itens mais importantes do *Product Backlog* e definidos quais deles irão integrar o *Sprint Backlog*.
- *Daily Scrum*: Reunião diária com duração de 15 minutos que visa monitorar o progresso da *sprint*, identificar impedimentos e fazer as devidas adaptações.
- *Sprint Review*: Reunião realizada ao fim de cada *sprint* onde a equipe apresenta os resultados gerados ao longo do período.
- *Sprint Retrospective*: Reunião que fecha a *sprint* e tem como finalidade entender o que funcionou e quais pontos podem ser melhorados, traçando possíveis melhorias para a próxima *sprint*.

2.1.1.3 Scrum Artifacts

Os *Scrum Artifacts* são informações utilizadas no Scrum com a finalidade de manter a transparência no processo, garantindo que toda a equipe tenha pleno acesso às informações.

- *Product Backlog*: É uma lista de itens nos quais os desenvolvedores irão trabalhar para o desenvolvimento do projeto.
- *Sprint Backlog*: Semelhante ao *Product Backlog*, porém abrange os itens que serão trabalhados durante a *sprint*.
- Incremento: É a soma das tarefas oriundas do *Product Backlog* prontas a cada *sprint*.

2.1.2 Front-end e back-end

De acordo com (COURSERA, 2020), ao seguir uma carreira em programação o profissional pode trabalhar no desenvolvimento *front-end*, *back-end* ou ambos.

O *front-end* é a área responsável pela interface do usuário, controlando os aspectos visuais e interativos de uma aplicação. Nesta área é importante prezar por interfaces responsivas, acessíveis e amigáveis, a fim de fornecer a melhor experiência possível para o usuário.

O *back-end* se encarrega da lógica principal de uma aplicação, sendo responsável pelo processamento e armazenamento dos dados, assim como a comunicação com outros sistemas ou serviços. É importante que um sistema *back-end* consiga responder às requisições com rapidez, garantindo a eficiência do serviço, porém sem negligenciar o cuidado com a segurança dos dados com os quais o sistema lida, oferecendo uma experiência segura e confiável aos usuários.

Em geral, os *back-ends* tornam-se acessíveis ao *front-end* por APIs (*Application Programming Interface*), que retornam e recebem dados por meio de requisições realizadas através da *internet*.

Na Figura 2.1, é possível visualizar um exemplo dessa comunicação, onde o *front-end* é responsável pelo que o usuário vê e interage, coletando dados e exibindo resultados, que são enviados ou recebidos do *back-end*, responsável por executar a lógica principal, além de se conectar ao banco de dados. Na figura também é possível observar que o *back-end* se conecta ao sistema de arquivos. Isso ocorre, pois algumas arquiteturas se baseiam no *Server Side Rendering*, que permite gerar o HTML da página no servidor e retorná-lo pronto ao navegador.

2.2 Ferramentas para desenvolvimento

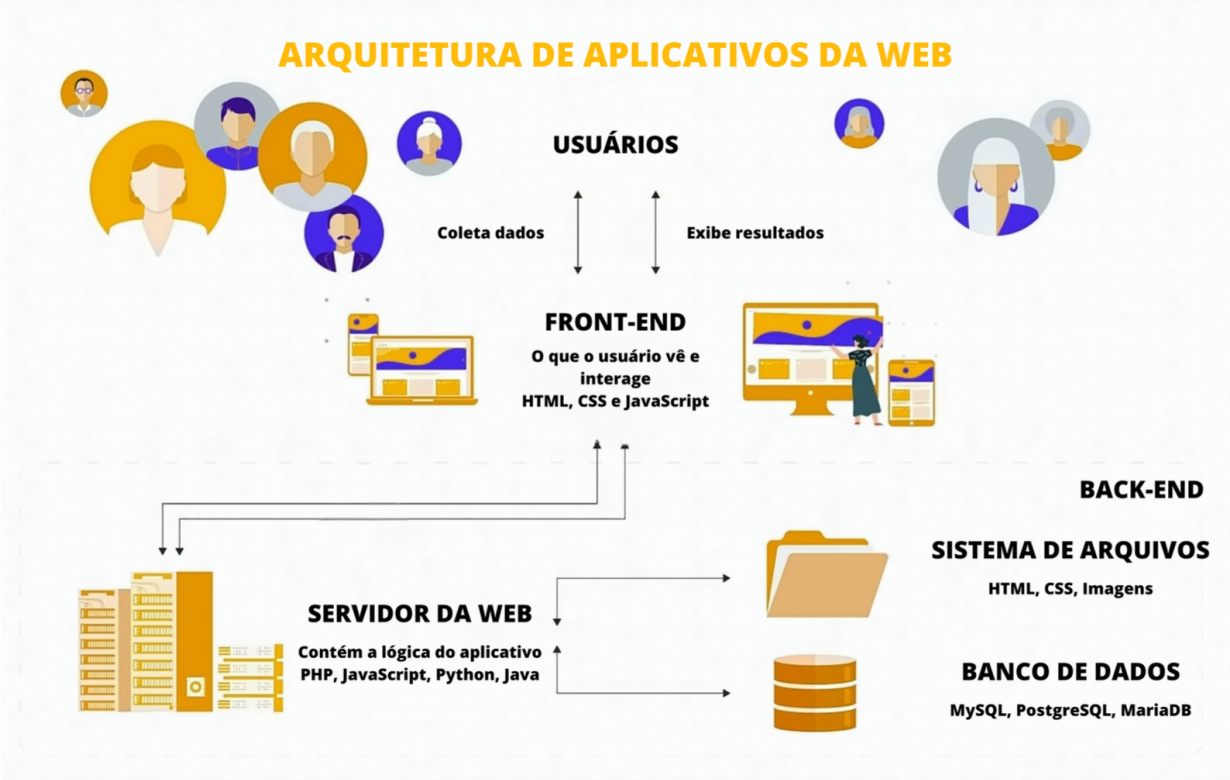
Nesta seção, são apresentadas as ferramentas utilizadas durante o desenvolvimento do projeto.

2.2.1 Visual Studio Code

O Visual Studio Code[1] é um editor de código desenvolvido pela Microsoft que conta com diversos recursos, como controle de versão Git incorporado, realce de sintaxe, suporte

[1] <https://code.visualstudio.com/>

Figura 2.1 – Exemplo de comunicação entre *front-end* e *back-end*



Fonte: (LEGIERSKI, 2021, tradução nossa)

para depuração, entre outros. É um *software* livre, cujo código-fonte está disponível sob a licença MIT. Além disso, possui uma ampla variedade de extensões que adicionam recursos adicionais à ferramenta, permitindo que seus usuários possam personalizá-lo de acordo com suas necessidades específicas.

No projeto que o aluno foi alocado, a utilização da ferramenta foi necessária, acompanhada de duas extensões: o Prettier[2], responsável por formatar o código e manter um padrão, e o Eslint[3], responsável por identificar problemas no código.

2.2.2 Git e GitLab

O Git[4] é um sistema de controle de versões com diversos recursos voltados para o gerenciamento do histórico de alterações em um projeto. Seu principal diferencial é o modelo de *branches* que adota, capaz de criar ramificações de um repositório e permitindo modificações sem alterar o código original.

[2] <https://prettier.io/>
[3] <https://eslint.org/>
[4] <https://git-scm.com/>

O GitLab[5] é um gerenciador de repositórios para projetos que utilizem o Git, possuindo diversas ferramentas auxiliares, como integração contínua, *pipeline* de entrega e testes. É muito utilizado como uma alternativa *open source* ao GitHub, por permitir que a ferramenta seja executada em servidores privados.

2.3 Tecnologias Utilizadas

Nesta seção, são apresentadas as linguagens de programação, *frameworks* e bibliotecas utilizadas durante o desenvolvimento do projeto.

2.3.1 HTML

O HTML (*Hypertext Markup Language*) é uma linguagem de marcação utilizada para a construção de páginas na *web*. Através de tags como *div*, *p* e *span* informa aos navegadores o que é cada elemento e onde devem aparecer. Atualmente se encontra em sua versão 5 e conta com diversos recursos como exibição de textos, *links*, imagens, vídeos e gráficos.

A estrutura gerada com o carregamento de um conteúdo HTML recebe o nome de DOM (*Document Object Model*), sendo este uma representação da página de uma forma que os programas possam alterar a estrutura, estilo e conteúdo do documento (MOZILLA FOUNDATION, 2022a). A estrutura do DOM pode ser representada como uma árvore de nós, onde cada nó corresponde a um elemento do documento HTML. O nó mais alto na árvore é o nó raiz, que representa todo o documento, e os nós abaixo dele representam os elementos, que podem conter outros elementos aninhados.

A Figura 2.2 possui um código HTML de exemplo e a Figura 2.3 apresenta uma visualização de sua árvore DOM, com seus elementos, atributos e textos.

2.3.2 CSS e SASS

O CSS (*Cascading Style Sheets*) é uma linguagem de estilo usada para definir a aparência de páginas *web*. Pode ser aplicado de várias formas, incluindo diretamente nas *tags* HTML,

[5] <https://about.gitlab.com/>

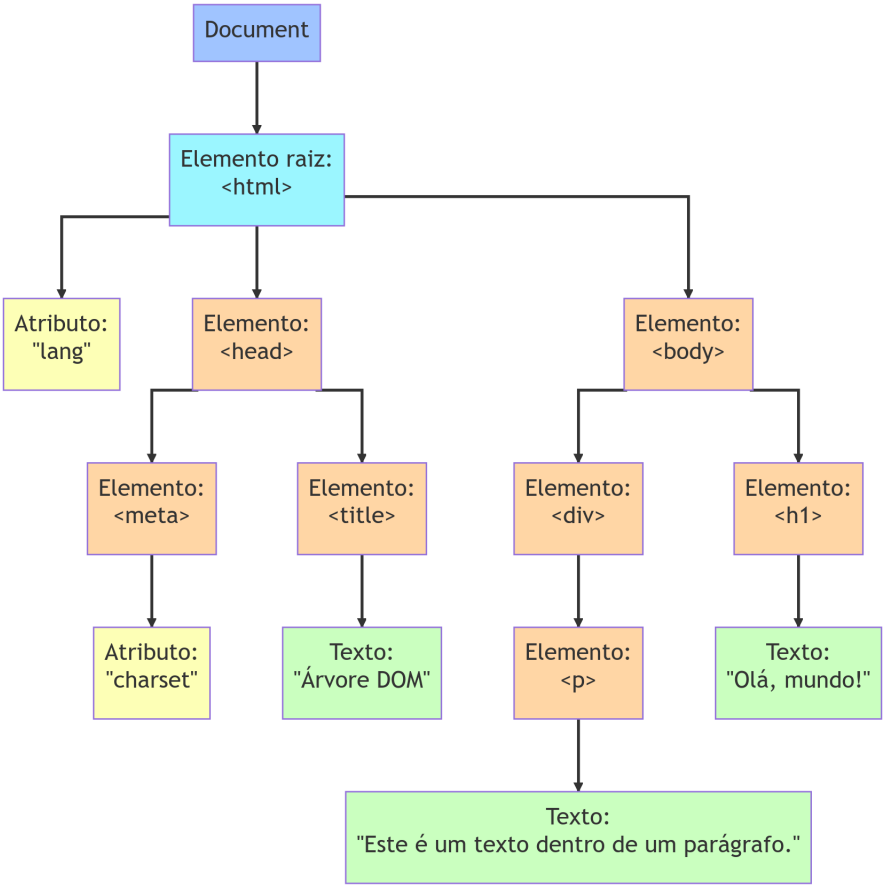
Figura 2.2 – Código de exemplo na linguagem HTML

```
<html lang="pt-BR">
  <head>
    <meta charset="UTF-8" />
    <title>Árvore DOM</title>
  </head>

  <body>
    <h1>Olá, mundo!</h1>
    <div>
      <p>Este é um texto dentro de um parágrafo.</p>
    </div>
  </body>
</html>
```

Fonte: Autor

Figura 2.3 – Visualização de uma árvore DOM



Fonte: Autor

em *tags* `<style>` ou em arquivos separados. Nos últimos dois casos, os elementos a serem estilizados são especificados usando seletores, como o nome da *tag*, classe ou *id* do elemento.

A linguagem oferece uma variedade de propriedades, como posição, cor, tamanho e efeitos de transição. Além disso, permite adaptar o estilo da página para diferentes tamanhos de tela e dispositivos.

O SASS (*Syntactically Awesome Style Sheets*) é um pré-processador CSS que adiciona recursos avançados, como variáveis, aninhamento de seletores e operações matemáticas. Ele foi criado para escrever CSS de forma mais robusta e sustentável, tornando-o menor, menos complexo e mais fácil de manter. Os arquivos SASS podem ser compilados em arquivos CSS, permitindo sua renderização em qualquer navegador.

A Figura 2.4 apresenta um exemplo de código em SASS (código à esquerda) e sua versão compilada em CSS (código à direita).

Figura 2.4 – Exemplo de código SASS e sua versão compilada em CSS

```scss
$color-primary: #3498db;

button {
  border: none;
  border-radius: 4px;
  color: white;
  font-size: 1rem;
  padding: 10px 20px;
  background-color: $color-primary;

  &:hover {
    background-color: darken($color-primary, 10%);
  }

  &:active {
    background-color: darken($color-primary, 20%);
  }
}
```

```css
button {
  border: none;
  border-radius: 4px;
  color: white;
  font-size: 1rem;
  padding: 10px 20px;
  background-color: #3498db;
}

button:hover {
  background-color: #217dbb;
}

button:active {
  background-color: #196090;
}
```

Fonte: Autor

2.3.3 JavaScript

O JavaScript é uma linguagem de programação interpretada versátil, que pode ser usada tanto em *scripts* para páginas *web* quanto em ambientes sem navegador, como o Node.js (MOZILLA FOUNDATION, 2022b). É amplamente utilizado na programação *web* para criar aplicações interativas e dinâmicas, permitindo a alteração do comportamento das páginas. Combinado com HTML e CSS, é considerado uma das três principais linguagens fundamentais para o desenvolvimento de aplicações *web*.

Atualmente, o JavaScript segue o padrão ECMAScript, sendo este uma especificação que estabelece os padrões para a linguagem, garantindo compatibilidade e interoperabilidade

entre diferentes implementações do JavaScript. Além disso, o padrão também é usado para adicionar novos recursos e melhorias à linguagem.

Ao transmitir e receber dados com JavaScript, é comum utilizar o JSON (*JavaScript Object Notation*), consistindo em um formato de texto que representa dados estruturados seguindo a sintaxe de objetos da linguagem.

Algumas bibliotecas permitem o desenvolvimento utilizando a sintaxe JSX, que possibilita a escrita de código HTML, CSS e JavaScript no mesmo arquivo. Há uma variação dessa sintaxe chamada TSX, que utiliza o TypeScript no lugar do JavaScript.

2.3.4 TypeScript

O TypeScript[6] é uma linguagem de programação baseada no JavaScript, desenvolvida para aumentar a escalabilidade e robustez dos projetos. Ele adiciona recursos como orientação a objetos e tipagem estática ao JavaScript, proporcionando um maior controle e previsibilidade no desenvolvimento de projetos.

Como uma extensão do JavaScript, a tecnologia herda os recursos da linguagem base e pode ser convertido para funcionar em todos os ambientes suportados pelo JavaScript.

A Figura 2.5 apresenta um exemplo de código escrito em TypeScript que utiliza os recursos citados anteriormente.

2.3.5 Node.js e NPM

O Node.js[7] é um *software* que permite a execução de código JavaScript de forma assíncrona, baseada em eventos e sem a necessidade de um navegador. Permite controlar várias conexões simultaneamente, porém de maneira diferente do modelo mais comum, onde *threads* são usadas para gerenciar essas conexões em paralelo (OPENJS FOUNDATION, 2015). O uso de *threads* pode se tornar complexo e resultar em situações em que duas ou mais unidades ficam impossibilitadas de prosseguir com suas operações, conhecidas como *deadlocks*.

O NPM[8] (*Node Package Manager*) é um gerenciador de pacotes utilizado para administrar projetos feitos em Node.js. Entre suas funcionalidades, destacam-se o registro de comandos

[6] <https://www.typescriptlang.org/>
[7] <https://nodejs.org/>
[8] <https://www.npmjs.com/>

Figura 2.5 – Código de exemplo na linguagem TypeScript

```
interface Usuario {
  nome: string;
  idade: number;
}

function exibirUsuario(usuario: Usuario): void {
  console.log(`Nome: ${usuario.nome}`);
  console.log(`Idade: ${usuario.idade}`);
}

const novoUsuario: Usuario = {
  nome: 'Fulano',
  idade: 30,
};

exibirUsuario(novoUsuario);
```

Fonte: Autor

personalizados e sua execução simplificada por meio de palavras-chave, fácil adição de novas dependências em um projeto e a instalação de dependências já existentes através de um único comando.

Ao utilizar o NPM em um projeto, é criado um arquivo `package.json` com todas as informações necessárias para sua execução, como nome do projeto, versão, dependências, *scripts*, arquivo principal, entre outras. O comando para criação de um novo projeto que utiliza o NPM pode ser visto a seguir. Junto dele foi adicionado o parâmetro `-y`, que inicia o projeto com as configurações padrão:

```
$ npm init -y
```

Após a execução do comando, um arquivo `package.json` é gerado (Figura 2.6). Por nenhuma opção ter sido selecionada, o nome do projeto é o mesmo da pasta em que foi criado.

Além disso, o NPM possibilita que os desenvolvedores criem e disponibilizem os próprios pacotes para que outras pessoas os utilizem. Com isso, há grande economia de tempo, pois os usuários podem aproveitar o trabalho de outras pessoas e construir suas aplicações a partir dele. Isso não apenas acelera o processo de desenvolvimento, mas também ajuda a garantir que as funcionalidades sejam consistentes e bem integradas. A plataforma está em constante crescimento e em abril de 2020 o NPM já contava com mais de 1,3 milhão de pacotes publicados (NASSRI, 2020).

Figura 2.6 – Estrutura básica de arquivo de configuração em projetos NPM

```
{
  "name": "meu-projeto",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Fonte: Autor

2.3.6 Web Components e Stencil

Web Components são um conjunto de tecnologias que permitem criar componentes reutilizáveis e personalizados para a web. Eles possibilitam que os desenvolvedores criem seus próprios elementos HTML personalizados, com comportamento e estilos específicos.

O Stencil[9] é um compilador voltado para a construção de *Web Components* que utiliza TypeScript, JSX e CSS.

A tecnologia se propõe a ser "agnóstica de *framework*", baseando-se no princípio de que o maior benefício dos *Web Components* é a liberdade de utilizá-los independente de *framework*. Segundo Stencil (2017b), um dos grandes desafios na implantação de um *design system*, coleção de componentes que busca padronizar a aparência em interfaces presentes em uma empresa ou produto, é a necessidade de que todas as equipes de desenvolvimento utilizem apenas um conjunto de tecnologias. Por conta disso, ao construir *Web Components* com o Stencil, as equipes que fazem uso desses componentes podem utilizar as tecnologias que funcionem melhor para elas.

Os *Web Components* desenvolvidos com o Stencil podem ser utilizados diretamente nos *frameworks*, assim como também é possível gerar versões exclusivas para cada um, que oferecem uma alta integração aos recursos disponíveis nos *frameworks*. Atualmente o Stencil consegue gerar componentes específicos para Ember, React e Vue.

A ferramenta fornece um guia de estilos visando padronizar a criação de componentes, contando com orientações sobre estrutura de arquivos, padrões de nomenclatura, boas práticas

[9] <https://stenciljs.com/>

no TypeScript e organização de código. Define também a ordem em que cada elemento do componente deve aparecer, seguindo o conceito de que o código-fonte deve ser organizado como um artigo de jornal, com o resumo de mais alto nível no topo, seguido por informações cada vez mais detalhadas abaixo (STENCIL, 2018).

Disponibiliza métodos de ciclo de vida do componente, ativados em determinados momentos de sua execução, como: toda vez que se conecta ao DOM, antes de ser renderizado, após alguma propriedade ser alterada, entre outros. Quando os critérios de ativação desses métodos são satisfeitos, todo código presente nos mesmos será executado. A Figura 2.7 apresenta em quais momentos esses métodos são ativados e sua ordem de execução.

Além disso, oferece testes *End-to-end* prontos para uso, que permite testes mais próximos do ambiente de produção, utilizando um navegador real. Conta ainda com um conjunto de funções que auxiliam a simular interações e verificar como o componente se comporta, bem como a geração de capturas de tela e comparação com capturas anteriores, informando possíveis diferenças na exibição do componente.

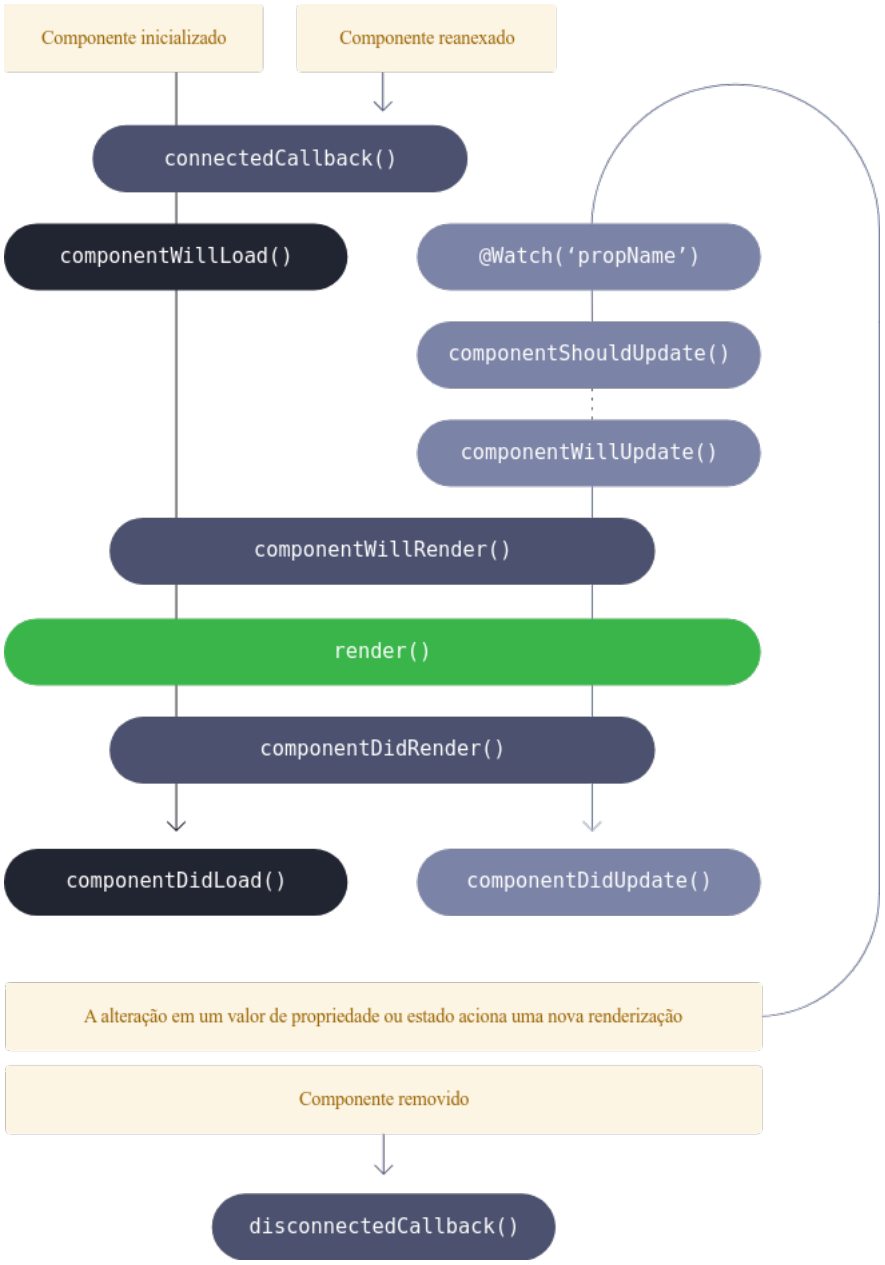
2.3.7 Storybook

O Storybook[10] é uma ferramenta que permite construir componentes de interface isoladamente. Visa simplificar a complexidade que o desenvolvimento *front-end* se tornou, com diversos componentes gerando milhares de variações. A medida que um projeto cresce, fica cada vez mais difícil a criação e depuração de componentes, levando a uma queda na qualidade dos projetos e gerando um comportamento imprevisível nas interfaces de usuário.

Com a ferramenta, é possível definir qualquer variação desejada para um componente, através da simulação dados e eventos, assim como a passagem de propriedades. Dessa forma, o desenvolvedor não precisa se preocupar em reproduzir cada fluxo possível que afete o componente, podendo focar exclusivamente na sua criação.

[10] <https://storybook.js.org/>

Figura 2.7 – Métodos de ciclo de vida do componente no Stencil



Fonte: (STENCIL, 2017a, tradução nossa)

2.3.8 Elasticsearch

O Elasticsearch[11] é uma ferramenta de busca e análise de dados altamente flexível e escalável, capaz de lidar com uma ampla variedade de tipos de dados, incluindo texto estruturados ou não estruturados, números e dados geoespaciais.

[11] <https://www.elastic.co/elasticsearch/>

A ferramenta possui mecanismos que armazenam e indexam os dados, possibilitando buscas rápidas. Além disso, o recurso de agregação de informações permite identificar tendências e padrões nos dados, tornando-a uma ferramenta valiosa para análise de dados.

2.3.9 SYDLE ONE e SYBOX

O SYDLE ONE, também chamado apenas de ONE, é uma solução *low-code* fundamentada em gestão de processos, gestão de conteúdo organizacional e *analytics*, focados na transformação digital das empresas. Além dessas ferramentas, são disponibilizados outros recursos como gestão de relacionamento com o cliente, gestão de cobranças de produtos, serviços e assinaturas, centro de serviços compartilhados, *e-commerce*, *service desk* e gestão de energia (SYDLE, 2023).

A plataforma permite que seus usuários modelem e documentem de forma gráfica todos os seus processos, centralizando-os digitalmente em um único local. Por conta dessa praticidade, é possível acompanhar cada processo individualmente, visualizando informações como quais etapas já foram concluídas e em qual se encontra.

Esse conjunto de soluções é denominado SYBOX e tem como vantagem sua fácil importação para um ambiente do ONE. Uma organização pode possuir diversos SYBOX integrados ao seu ambiente, conforme suas necessidades operacionais.

2.3.10 SYDLE UI

A SYDLE UI é um conjunto de soluções pensadas para facilitar o desenvolvimento *front-end* nos projetos da SYDLE e conta com recursos como ferramentas de hospedagem e gestão de aplicações através do SYDLE ONE, bibliotecas de componentes e incentivo ao compartilhamento de recursos.

O SYBOX da SYDLE UI possibilita o registro e gerenciamento de aplicações e suas dependências por objetos do ONE, facilitando o desenvolvimento de *front-ends* que o possuam como *back-end*.

Também oferece bibliotecas de componentes que atuam como um *design system* para a SYDLE e possibilitam a inserção de componentes prontos, permitindo um maior foco dos desenvolvedores nas regras de negócio. Por serem desenvolvidas em Stencil, também buscam

ser “agnósticas de *frameworks*”, dando liberdade ao usuário para as utilizarem onde desejar. Sua principal biblioteca também se chama SYDLE UI, possuindo dezenas de componentes com diversas opções de customização, como botões, *inputs*, alertas, *dialogs*, entre outros.

Uma vantagem significativa é que além do time dedicado ao desenvolvimento da SYDLE UI e suas bibliotecas, há um incentivo para que outros membros das equipes que utilizam essas ferramentas possam contribuir com suas próprias melhorias, corrigindo *bugs* e adicionando novas funcionalidades conforme suas necessidades específicas.

2.3.11 Service Desk Components

De acordo com SYDLE (2020), *service desk* é uma central de serviços de TI que oferece suporte técnico especializado para o cliente que contata uma empresa, promovendo uma comunicação eficiente através da centralização dos pontos de contato. Oferece recursos como solicitações de suporte e seu histórico completo, além de gerenciar diversos processos que mantêm a empresa funcionando através da padronização de tarefas e fluxos de trabalho. A Figura 2.8 apresenta os principais benefícios em utilizar um *service desk*.

A SYDLE possui um time responsável por desenvolver soluções reaproveitáveis utilizando as funcionalidades do SYDLE ONE e disponibilizá-las no formato de SYBOX. Este time é dividido em equipes, onde cada equipe é responsável por uma solução, sendo o *service desk* uma delas.

O Service Desk Components utiliza os componentes básicos oferecidos pela biblioteca SYDLE UI, mantendo assim a identidade visual da SYDLE. Usa também o SYDLE ONE Components, biblioteca que possibilita o uso de componentes do SYDLE ONE no projeto e oferece funções auxiliares para a comunicação com a API do ONE.

As atividades referentes ao projeto se encontram no sistema interno da empresa e são exibidos em um quadro virtual semelhante ao apresentado na Figura 2.9, apresentando cartões que podem ser abertos para revelar mais detalhes. Suas principais informações são:

- Código: Código único que distingue cada tarefa, apresentando o formato MLH123456 ou BUG123456. As tarefas começadas em MLH são melhorias e as começadas em BUG são *bugs*, os números após essas letras são sequências geradas para cada tarefa.

- Título: Uma breve descrição do que deve ser efetuado na tarefa.

Figura 2.8 – Principais benefícios no uso de um *service desk*

Fonte: (SYDLE, 2020)

- Detalhes: Uma descrição detalhada da tarefa e suas possíveis soluções.
- Prioridade: Define o nível de urgência e importância da tarefa, podendo ser Alta, Média ou Baixa.
- Status: Determina em que fase se encontra a tarefa.

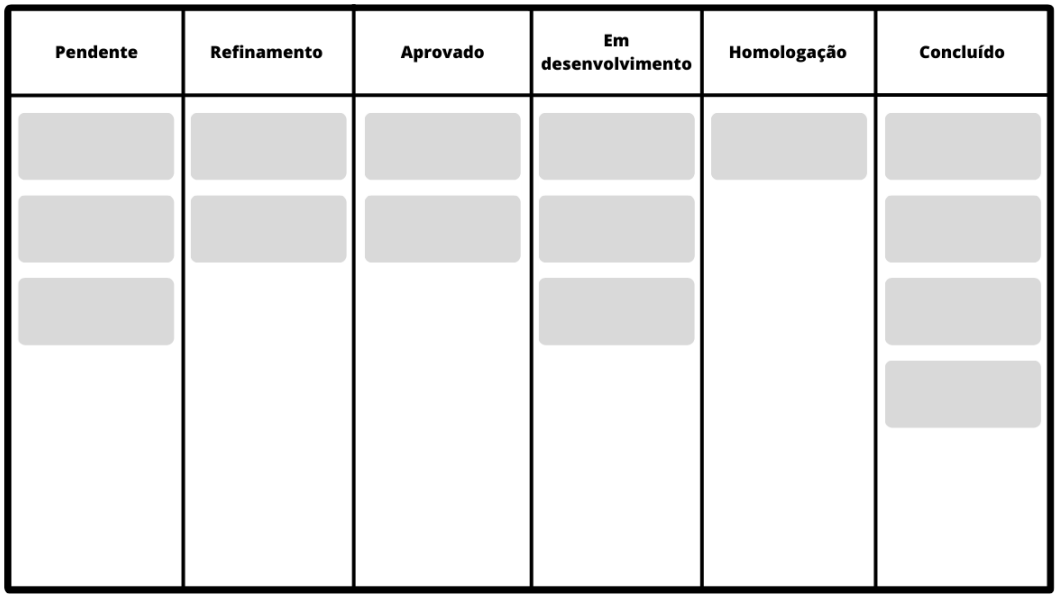
As possíveis fases para uma tarefa são exibidas no quadro como colunas, que podem ser:

- Pendente: Tarefas recém-criadas. Surgem por necessidade de novos recursos ou *bugs* reportados por usuários e membros da equipe. Sua descrição costuma conter a mensagem enviada pela pessoa que solicitou a funcionalidade ou reportou o *bug*.
- Refinamento: Tarefas que serão realizadas na próxima *sprint*. O líder técnico faz uma análise buscando entender quais serão os passos necessários para a realização da tarefa. Sua descrição contém orientações sobre quais mudanças podem ser necessárias, como os componentes que devem ser alterados no *front-end* e/ou quais classes, métodos e campos

devem ser criados ou alterados no ONE. As informações presentes na descrição funcionam como um ponto de partida ao membro que irá executar a tarefa, podendo ser necessários mais passos para sua conclusão.

- Aprovado: Tarefas que serão realizadas na *sprint* atual. As tarefas presentes nessa coluna podem ser escolhidas por qualquer membro da equipe.
- Em desenvolvimento: Tarefas sendo realizadas pelos membros no momento.
- Homologação: Tarefas finalizadas, que aguardam revisão da equipe.
- Concluído: Tarefas revisadas. Coluna limpa ao fim de cada *sprint*.

Figura 2.9 – Exemplo simplificado do quadro de tarefas do projeto



Fonte: Autor

Após a escolha de uma tarefa, é necessária a criação de uma nova *branch*, a partir de outra que contém as últimas modificações revisadas pela equipe, denominada *develop*.

Quando a tarefa é completa, uma solicitação para incluir as alterações realizadas na *develop* é aberta no GitLab do projeto, detalhando a qual tarefa se refere e quais alterações foram feitas. As mudanças são revisadas por todos os membros da equipe, garantindo que os integrantes mais antigos identifiquem possíveis erros e os mais recentes se familiarizem com o projeto. Após essa revisão, as mudanças já estão aptas a serem adicionadas à *develop*.

3 ATIVIDADES DESENVOLVIDAS

Neste capítulo, são detalhadas as principais atividades desempenhadas pelo aluno como estagiário da empresa SYDLE no período de 04/07/2022 a 10/02/2023. Foram realizados treinamentos sobre os sistemas internos da empresa e as principais tecnologias utilizadas pela equipe na qual o aluno foi alocado. Além disso, foram desempenhadas tarefas de melhorias, correção de bugs, criação de componentes e construção de ferramentas para auxílio no processo de desenvolvimento.

3.1 Período de treinamento

A primeira semana de todo ingressante na SYDLE é reservada para uma apresentação da empresa e um treinamento sobre o SYDLE ONE.

O treinamento tem duração de cinco dias, possuindo uma aula ao vivo no início de cada dia, seguido de conteúdo teórico e prático.

O sistema interno da empresa oferece todos os recursos necessários para o estudo, incluindo vídeo-aulas e a Trilha Foundations, uma série de tutoriais que unem teoria com atividades práticas, sendo organizado da seguinte forma:

- Dia 1: No primeiro dia, é realizada a apresentação do ONE e de sua abordagem para ECM (Gestão Eletrônica de Conteúdo). Segundo SYDLE (2023), ECM é a gestão integrada de conteúdo combinada com processos para gerar e manter grandes volumes de dados estruturados, atualizados e confiáveis. Neste dia, o colaborador adquire uma compreensão geral do ONE, compreendendo sua natureza, propósito, funcionamento e amplo escopo de utilização. Além disso, é apreendido sobre modelagem de dados e de que forma é realizada no contexto do ONE.

- Dia 2: No segundo dia, é explorado o BPM (Gestão por Processos de Negócio), que procura processos automatizados, flexíveis e orientados a resultados para negócios escaláveis de alta qualidade (SYDLE, 2023). Neste dia, é adquirido conhecimento sobre conceitos amplamente utilizados na rotina da SYDLE, como o que são processos, o funcionamento da gestão por processos e a notação BPMN.

- Dia 3: No terceiro dia, o assunto tratado são as automações e integrações, recursos do SYDLE ONE que trazem uma maior inteligência aos processos através de *scripts* perso-

nalizados e APIs. Neste dia, é ensinado o que são os métodos padrões, presentes em todas as classes, e como utilizá-los para alcançar o comportamento desejado na modelagem de processos. Também é ensinado como criar métodos customizados e sua utilização por APIs ou da interface do ONE.

- Dia 4: No quarto dia, foi apresentado o Elasticsearch, além dos conceitos de SYBOX. Neste dia foi ensinado sobre as consultas do Elasticsearch nos métodos de uma classe e como fazer a importação e exportação entre ambientes utilizando o SYBOX.

- Dia 5: No último dia, foram passados exercícios extras para fixar os conteúdos do dia anterior. Também foi dada uma tarefa guiada para a criação de um processo de solicitação de compra, utilizado em empresas para que seus funcionários requisitem materiais necessários em sua área de trabalho. A criação deste processo foi uma tarefa extensa que permitiu a revisão de diversos conceitos aprendidos ao longo do treinamento.

Após o treinamento sobre o ONE, o aluno foi alocado no time responsável pelo *service desk*, cujo líder técnico instruiu a realizar o estudo de um guia intitulado "*Roadmap* de aprendizado", focado nos novos membros dos times do SYBOX.

O guia apresentava uma introdução aos conhecimentos necessários para o desenvolvimento do projeto, com um vídeo introdutório e um material textual para a maioria dos tópicos. Os tópicos vistos foram: HTML, CSS, SASS, JavaScript, TypeScript, NPM, Web Components, Stencil e Git. Após passar por todos esses tópicos, o aluno estava apto para executar tarefas no projeto.

3.2 Tarefas realizadas

O projeto desenvolvido pela equipe a qual o aluno pertence se chama Service Desk Components, que migra o portal de serviços utilizado até o momento pela SYDLE e seus clientes, construído em uma tecnologia que não seguia os padrões adotados pela empresa. Além disso, o projeto adicionou novos recursos, que serão apresentados mais adiante.

A equipe utilizou uma versão adaptada do Scrum, em que cada *sprint* do projeto dura aproximadamente um mês, com uma reunião de planejamento no início da *sprint*, além de reuniões diárias.

O projeto é desenvolvido em Stencil, utilizando TypeScript com a sintaxe TSX para a lógica e estruturação dos componentes, assim como o SASS para a estilização. Além disso, o projeto conta com testes *End-to-end* e o Storybook, que permite uma visualização isolada dos componentes.

A seguir, serão citadas algumas das tarefas realizadas pelo aluno que mais contribuíram para seu aprendizado acadêmico e profissional.

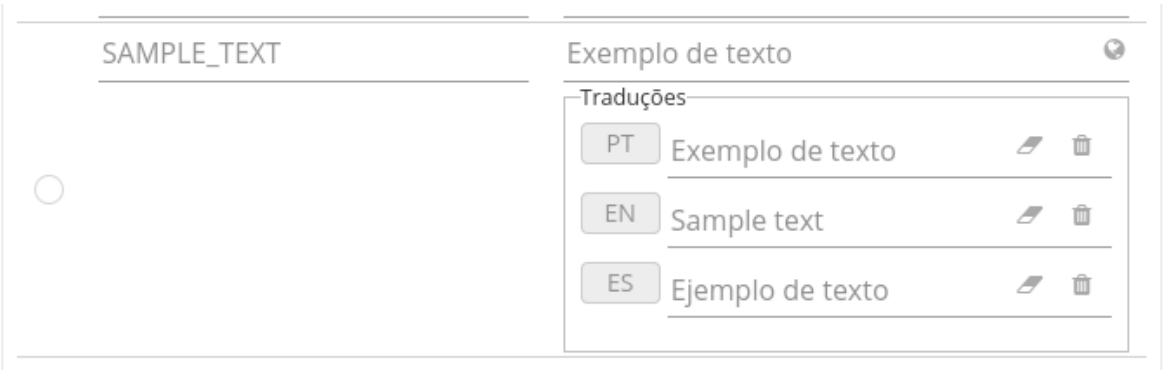
3.2.1 Refatoração dos componentes

Ao entrar na equipe, o aluno foi orientado a realizar tarefas de refatoração, que surgiram com o amadurecimento do projeto e definição de novos padrões a serem seguidos. A tarefa foi dividida entre os membros da equipe, que refatoravam um componente de cada vez. O aluno executou os passos detalhados a seguir para cada componente refatorado.

Os componentes do projeto não seguiam completamente o guia de estilos do Stencil, possuindo elementos fora de ordem que dificultavam a leitura do código. Sendo assim, cada componente foi verificado e ajustado conforme o guia, caso não respeitasse o padrão.

O SYDLE ONE possui uma classe chamada “Pacote de tradução”, que possibilita o cadastro de traduções, onde o usuário cria uma chave e as traduções nos idiomas desejados (Figura 3.1). A classe permite chamada via API, recebendo qual o idioma e pacote de tradução desejado e retornando as chaves cadastradas e seus respectivos valores.

Figura 3.1 – Exemplo de chave de tradução em um Pacote de tradução no SYDLE ONE



Fonte: Autor

A biblioteca SYDLE ONE Components, utilizada pelo Service Desk Components, oferece um componente responsável por simplificar a comunicação com o pacote de tradução, bas-

tando adicioná-lo na raiz do projeto passando como propriedade o identificador e *id* do pacote que deverá ser carregado.

Com o componente configurado, é possível obter as traduções cadastradas por uma função que recebe o identificador de algum pacote carregado, a chave desejada e um valor padrão que será exibido caso haja algum erro na obtenção do pacote.

Por conta do projeto utilizar apenas um pacote de tradução, uma função auxiliar foi criada para eliminar a necessidade de informar o identificador do pacote sempre que uma tradução fosse necessária[1].

No momento da refatoração, o pacote de tradução estava sendo incluído no projeto, porém haviam alguns componentes que ainda não o utilizavam. Por conta disso, todo texto exibido para o usuário que ainda fosse definido diretamente no *front-end* do projeto, assim como suas traduções, foi cadastrado no ONE e seus usos substituídos pela função de tradução utilizando a chave cadastrada para o texto.

Ao adicionar uma tradução, o aluno precisou se atentar ao local correto para essa adição, garantindo que seu carregamento ocorresse no momento certo. Na documentação do Stencil é possível encontrar a seguinte definição para o método `componentWillLoad`:

> Chamado uma vez logo após o componente ser conectado ao DOM. Como esse método é chamado apenas uma vez, é um bom lugar para carregar dados de forma assíncrona e configurar o estado sem acionar novas renderizações extras.[2] (STENCIL, 2017a, tradução nossa)

Por conta disso, esse método se torna o melhor local para o preenchimento das variáveis que dependem da tradução.

Ficou definido pela equipe que todo componente que necessitasse de tradução possuiria uma função chamada `setLabels`, responsável por preencher as traduções e centralizá-las como propriedades de uma variável chamada `translationBundle`. Essa função, quando existente, deveria ser chamada no `componentWillLoad` do componente, conforme exemplificado na Figura 3.2.

Na refatoração, o alunou implementou essa estrutura caso adicionasse traduções ou identificasse alguma fora do padrão.

---

[1] No contexto do projeto, tradução refere-se a qualquer texto exibido por um componente no idioma selecionado pelo usuário.

[2] "Called once just after the component is first connected to the DOM. Since this method is only called once, it's a good place to load data asynchronously and to setup the state without triggering extra re-renders."

Figura 3.2 – Exemplo de preenchimento de tradução

```
componentWillLoad(): void {
  this.setLabels();
}

private setLabels(): void {
  this.translationBundle = {
    sampleText: translation('SAMPLE_TEXT', 'Sample text'),
  };
}
```

Fonte: Autor

A biblioteca SYDLE UI passou a oferecer funções auxiliares para simplificar a criação de testes em projetos que a utilizam. Por conta disso, uma das mudanças necessárias na refatoração foi adequar os testes existentes para esse novo padrão. Com a mudança, novas capturas de tela foram geradas, substituindo as existentes.

Para a execução desta tarefa foram gastos dez dias de trabalho e o aluno realizou a refatoração de cinco componentes, levando um tempo médio de dois dias de trabalho por componente. Foram modificados 23 arquivos, com 293 linhas adicionadas, 174 linhas removidas e 260 linhas modificadas, além de 44 arquivos adicionados e 35 deletados, referentes às capturas de tela.

Essa tarefa possibilitou um aumento significativo na organização e legibilidade do código, tornando o desenvolvimento mais fluido e diminuindo o tempo gasto na análise ao se trabalhar com um componente existente.

Durante a execução, o aluno pôde adquirir conhecimentos fundamentais para sua evolução no projeto, sobretudo no desenvolvimento *front-end*, como boas práticas do Stencil, ciclo de vida dos componentes e criação de testes.

3.2.2 Componente de gráfico

No Service Desk, serviços são páginas exibidas ao usuário. Podem conter textos, formulários, *scripts* ou serem dinâmicas, com comportamento variável baseados em parâmetros definidos em sua construção. Um dos recursos disponíveis para os serviços dinâmicos são os componentes, que oferecem módulos de exibição de conteúdos pré-determinados.

Foi atribuída ao aluno uma tarefa referente à criação de um componente de gráfico para utilização nos serviços dinâmicos. O componente deveria ser criado utilizando a biblioteca

Chart.js[3], aceitando como propriedade um JSON responsável por definir os parâmetros do gráfico. Além disso, o gráfico gerado deveria se ajustar ao tamanho das telas, podendo ser visualizado em dispositivos menores, como celular. Ao fim da criação, a documentação dos componentes para serviço dinâmico deveria ser atualizada, possuindo um tutorial exemplificando o uso do gráfico.

Foi gerada a estrutura básica de um componente no Stencil através do comando `generate`, como mostrado abaixo.

```
$ stencil generate chart-widget
```

Durante a execução, foi exibida uma mensagem perguntando quais arquivos adicionais deveriam ser gerados. Entre as opções, apenas o arquivo de testes *End-to-end* foi selecionado, já que a opção de estilização (CSS) não é utilizada no projeto. O arquivo TSX gerado pode ser visto na Figura 3.3.

Figura 3.3 – Estrutura básica de um componente gerado pelo Stencil

```
import { Component, Host, h } from '@stencil/core';

@Component({
  tag: 'chart-widget',
  shadow: true,
})
export class ChartWidget {

  render() {
    return (
      <Host>
        <slot></slot>
      </Host>
    );
  }

}
```

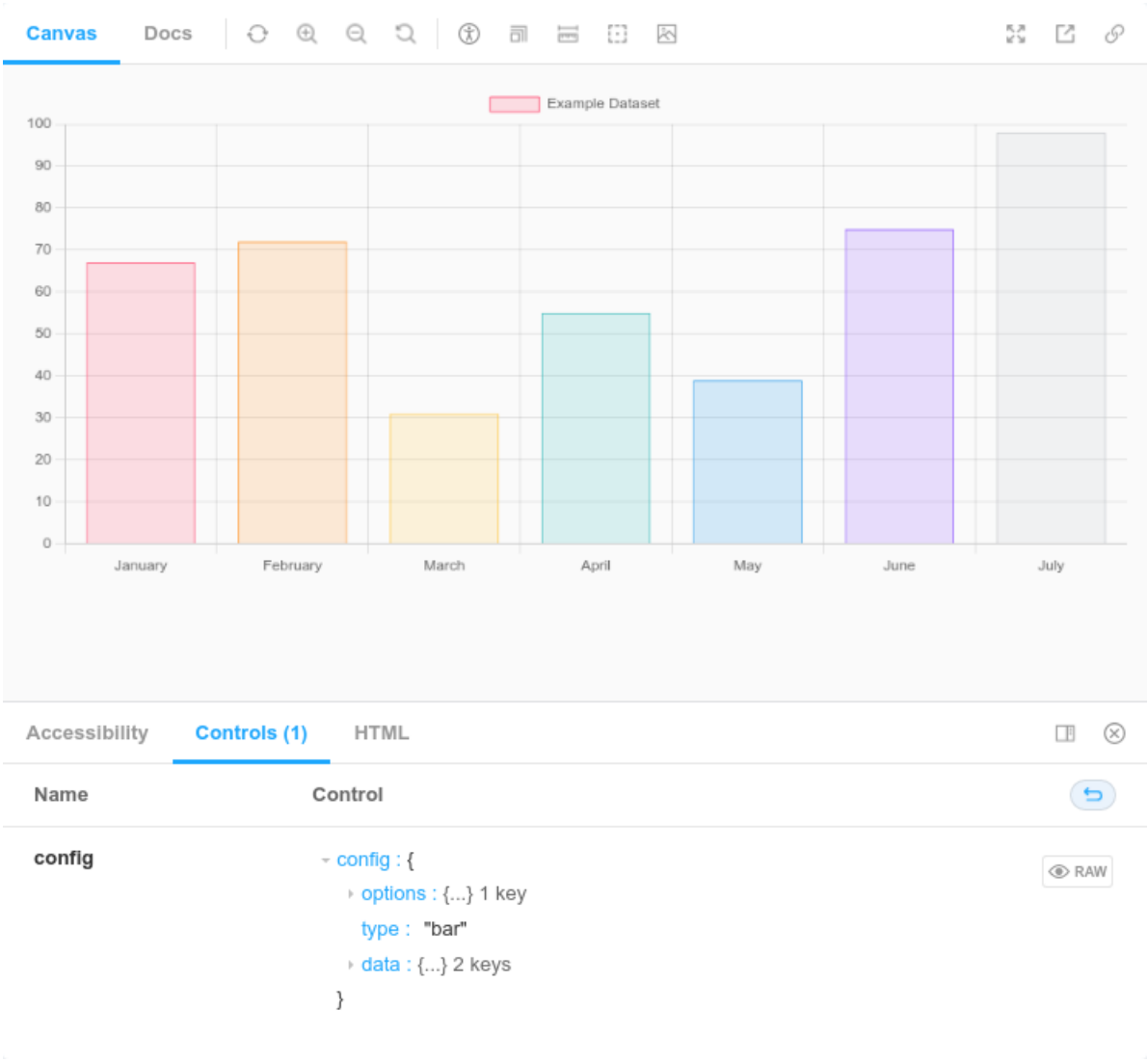
Fonte: Autor

A biblioteca Chart.js foi instalada no projeto e adicionada ao componente recém-criado, que carregava o gráfico conforme a configuração passada como propriedade. Não foram necessárias alterações para possibilitar a visualização em dispositivos menores, pois este é o comportamento padrão da biblioteca.

[3] <https://www.chartjs.org/>

Uma vez criado o componente, foram adicionados os testes, os quais verificavam se, carregando uma determinada configuração, o gráfico renderizava corretamente. Também foi feita sua adição ao Storybook (Figura 3.4), que exibia o componente nessas mesmas condições.

Figura 3.4 – Visualização do componente de gráfico no Storybook



Fonte: Autor

Com o *front-end* finalizado, o próximo passo foi configurar o uso do componente no SYDLE ONE. Para isso foi criada a classe de configuração para o gráfico, responsável por definir o *script* padrão de obtenção de dados. O *script* criado possuía apenas comentários, contendo o *link* para a documentação do Chart.js e uma configuração de exemplo.

Foi criada também a classe Gráfico, responsável por referenciar a classe de configuração e definir o método `Desenha`, que executa o *script* de obtenção de dados, encaminhando seu retorno para o componente criado no *front-end*. A classe criada passou a ser exibida como uma das opções de componente em um serviço dinâmico, e assim estava pronta para uso.

Por fim, a documentação para o componente foi adicionada, com o passo-a-passo de sua adição a um serviço dinâmico.

Para a execução desta tarefa foram gastos cinco dias de trabalho, sendo três deles para o *front-end* e o restante nas configurações do ONE. Foram modificados três arquivos, com 298 linhas adicionadas e 31 linhas modificadas, além de 12 arquivos adicionados, referentes às capturas de tela.

Essa tarefa agregou valor ao projeto, oferecendo ao usuários do produto uma nova forma de transmitir informações, exibindo dados de forma visual e intuitiva.

Durante a execução, o aluno pôde se aprofundar nos conceitos vistos anteriormente no treinamento do SYDLE ONE, como a criação de classes e métodos.

3.2.3 Script para geração de componentes

Ao realizar tarefas que exigiam a criação de novos componentes, o aluno percebeu que o comando `generate` fornecido pelo Stencil não atendia às necessidades do projeto. Além disso, a estrutura de arquivos era complexa, sendo necessário replicá-la a cada novo componente.

A partir dessa observação, foi feita uma análise buscando identificar os trechos de código que se repetiam. Com os trechos repetitivos localizados, um *script* para geração de componentes para o projeto começou a ser desenvolvido pelo aluno.

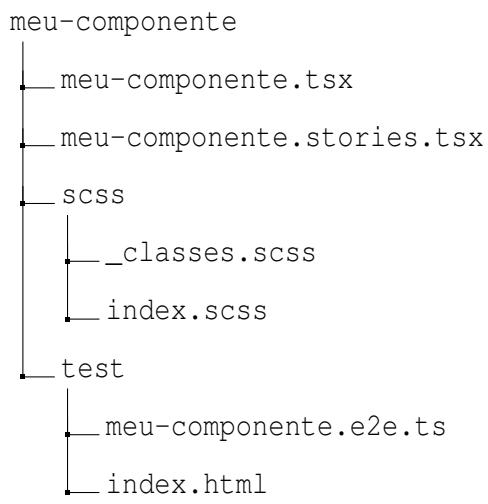
O *script* recebeu o nome de `generate-component` e possuía como único argumento o nome do componente a ser criado no formato *kebab-case*, onde cada palavra é separada por hífen, além de possuir apenas letras minúsculas. Foi adicionado no arquivo `package.json` um novo *script* apontando para o arquivo criado, podendo ser executado como exemplificado a seguir:

```
$ npm run generate <nome-do-componente>
```

O *script* possuía uma função principal, que fazia validações, como verificar se algum argumento foi passado e se o componente já existia. Em seguida chamava quatro funções, cada uma responsável por um aspecto do componente. São elas: criação do arquivo principal, Storybook, estilos e testes.

Cada função criava o arquivo em sua localização correta e ao fim da execução do *script*, a estrutura de pastas presente na Figura 3.5 era gerada.

Figura 3.5 – Estrutura de pastas gerada pelo *script* para geração de componentes



Fonte: Autor

Para a execução desta tarefa foi gasto um dia de trabalho. Foram modificados dois arquivos, com quatro linhas modificadas, além da criação do arquivo contendo o *script*, possuindo 135 linhas.

Essa tarefa trouxe economia de tempo para a equipe, eliminando o retrabalho que ocorria a cada componente criado e permitindo o foco em outras atividades de maior valor agregado para o projeto.

Durante a execução dessa tarefa, o aluno pôde compreender melhor a estrutura dos componentes, se aprofundando nos elementos fundamentais de cada arquivo e suas respectivas importâncias.

3.3 Visão geral do estágio

Após seis meses de estágio, o líder da equipe realizou uma reunião para avaliação do desempenho do aluno no projeto. Diversos pontos positivos observados no aluno foram citados durante a reunião, sendo os principais:

- O tempo gasto na realização das tarefas, que correspondia ou era menor que o esperado;
- Sua vontade em se desafiar, escolhendo tarefas com maior grau de dificuldade;
- Autogestão, adquirindo maior independência no decorrer do tempo.

Também foi citada a insegurança como um ponto a ser melhorado, porém, foi mencionada uma evolução recente neste quesito.

Durante o período de elaboração deste relatório, o aluno se encontra estagiando na empresa, que demonstra estar satisfeita com o trabalho realizado.

4 CONSIDERAÇÕES FINAIS

O período do aluno como estagiário na SYDLE proporcionou grande aprendizado técnico e aprimoramento nas habilidades interpessoais, aprofundando conhecimentos vistos durante o curso em disciplinas e outras atividades acadêmicas.

Ao atuar no projeto, o aluno precisou utilizar tecnologias que abrangiam diversas áreas do desenvolvimento de *software*. Por conta disso, foi necessário um aprimoramento contínuo, através do conhecimento passado por outros membros da equipe e pesquisas, com foco na leitura da documentação dessas tecnologias.

Ao decorrer do processo, surgiram impedimentos que não podiam ser resolvidos apenas com o conhecimento individual do aluno, gerando necessidade de comunicação constante com os demais envolvidos no projeto. Com o tempo, essa comunicação se tornou cada vez mais assertiva, possibilitando uma evolução na compreensão e transmissão de ideias.

Além disso, a cada tarefa o aluno adquiria uma maior autonomia, ganhando confiança no trabalho desempenhado e aumentando a qualidade de suas entregas.

Diversas disciplinas cursadas foram fundamentais para um bom aproveitamento do estágio, como, por exemplo: Introdução aos Algoritmos, Práticas de Programação Orientada a Objetos, Engenharia de *Software*, Interação Humano-Computador, entre outras.

Além disso, a participação em atividades extracurriculares forneceram conceitos que complementaram os já vistos em sala de aula. Como na Comp Júnior, uma empresa júnior que introduziu noções de mercado de trabalho e desenvolvimento *web* e no NESCAU (Núcleo de Estudos em Segurança Computacional e Auditoria), um núcleo de estudos que apresentou conceitos de segurança, como o desenvolvimento seguro, aplicados pelo aluno no projeto trabalhado.

Desse modo, conclui-se que o período de estágio foi fundamental para a formação pessoal, acadêmica e profissional do aluno, contribuindo para a construção de um profissional mais completo e preparado para os desafios da carreira. Assim, o estágio na SYDLE foi um marco importante na sua formação, consolidando seu aprendizado e tornando-o mais apto para uma atuação bem-sucedida no mercado de trabalho.

REFERÊNCIAS

COURSERA. **Front End vs. Back End: Learning Skills and Tools**. 2020. Disponível em: <https://www.coursera.org/articles/front-end-vs-back-end>. Acesso em: 06 fev. 2023.

LEGIERSKI, B. **Frontend vs Backend**. 2021. Disponível em: <https://www.evertop.pl/en/frontend-vs-backend/>. Acesso em: 07 fev. 2023.

MOZILLA FOUNDATION. **Document Object Model (DOM)**. [S.l.], 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/API/Document_Object_Model>. Acesso em: 17 fev. 2023.

MOZILLA FOUNDATION. **What is JavaScript?** [S.l.], 2022. Disponível em: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript>. Acesso em: 17 fev. 2023.

NASSRI, A. **npm Blog Archive: So long, and thanks for all the packages!** 2020. Disponível em: <https://blog.npmjs.org/post/615388323067854848/so-long-and-thanks-for-all-the-packages.html>. Acesso em: 06 fev. 2023.

OPENJS FOUNDATION. **About**. [S.l.], 2015. Disponível em: <https://nodejs.org/en/about/>. Acesso em: 17 fev. 2023.

SCHWABER, K.; SUTHERLAND, J. **The Scrum Guide**: The definitive guide to scrum: The rules of the game. [s.n.], 2020. Disponível em: <https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf>.

STENCIL. **Component Lifecycle Methods**. [S.l.], 2017. Disponível em: <https://stenciljs.com/docs/component-lifecycle>. Acesso em: 16 fev. 2023.

STENCIL. **Stencil - A Compiler for Web Components**. [S.l.], 2017. Disponível em: <https://stenciljs.com/docs/introduction>. Acesso em: 18 fev. 2023.

STENCIL. **Stencil Style Guide**. [S.l.], 2018. Disponível em: <https://stenciljs.com/docs/style-guide>. Acesso em: 16 fev. 2023.

SYDLE. **O que é Service Desk? Como diferenciar de help desk?** 2020. Disponível em: <https://www.sydle.com/br/blog/service-desk-o-que-e-5f5be047c053d93b8e3b68ae/>. Acesso em: 16 fev. 2023.

SYDLE. **SYDLE ONE**. 2023. Disponível em: <https://www.sydle.com/br/>. Acesso em: 16 fev. 2023.