PEDRO ANTÔNIO DE SOUZA

RELATÓRIO DE ESTÁGIO:

DESENVOLVIMENTO *LOW-CODE* NA PLATAFORMA SYDLE ONE

LAVRAS – MG

2022

PEDRO ANTÔNIO DE SOUZA

RELATÓRIO DE ESTÁGIO:
DESENVOLVIMENTO *LOW-CODE* NA PLATAFORMA SYDLE ONE

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

Prof. DSc. Raphael Winckler de Bettio
Orientador

LAVRAS – MG

2022

PEDRO ANTÔNIO DE SOUZA

RELATÓRIO DE ESTÁGIO: DESENVOLVIMENTO *LOW-CODE* NA PLATAFORMA SYDLE ONE

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

APROVADA em 8 de setembro de 2022.

Prof. DSc. André Grützmann	UFLA
Prof. DSc. Maurício Ronny de Almeida Souza	UFLA

gov.br
Documento assinado digitalmente
RAPHAEL WINCKLER DE BETTIO
Data: 16/09/2022 15:45:32-0300
Verifique em https://verificador.iti.br

Prof. DSc. Raphael Winckler de Bettio
Orientador

LAVRAS – MG
2022

AGRADECIMENTOS

Agradeço, especialmente, à Elisângela e ao Marcelo, meus pais, e à Ana, minha irmã, por me proporcionarem um ambiente saudável, de carinho e apoio nas minhas decisões.
À minha companheira, Gabriela, por todo amor dado a mim e pelas palavras de motivação nos momentos que precisei.
Aos meus colegas e professores, pelos ensinamentos e pela atenção que me deram durante a jornada de aprendizagem.
Ao professor DSc. Raphael Winckler de Bettio, deixo um agradecimento especial pelas orientações na escrita do presente trabalho e no programa de atividade vivencial.
À SYDLE por ter me dado a oportunidade de iniciar minha carreira profissional em uma empresa respeitável com ambiente descontraído e formador de amizades.
Aos meus colegas de equipe, agradeço pelas importantes conquistas e pelos bons momentos que vivemos.

A alegria não chega apenas no encontro do achado, mas faz parte do processo da busca.
(Paulo Freire)

RESUMO

Este documento relata as atividades realizadas pelo autor durante seu estágio em desenvolvimento de *software* na empresa SYDLE. Sediada em Belo Horizonte, a empresa possui como produto principal o SYDLE ONE: uma plataforma *low-code* que engloba diversas soluções corporativas. Para se tornar apto ao desenvolvimento no SYDLE ONE, o estagiário participou de um treinamento que apresentou a plataforma e introduziu as tecnologias Elasticsearch e BPMN, fundamentais para atuar em projetos da empresa. Após o treinamento, o estagiário foi alocado, por 10 meses, ao projeto de certificação digital contratado por uma empresa privada. Esse projeto objetivava fornecer soluções de ponta a ponta para modernizar a atuação da contratante na venda e emissão de certificados digitais. Dessa forma, a equipe era responsável desde a manutenção do *e-commerce* até os processos de emissão e instalação dos documentos eletrônicos. Além de aprimorar suas capacidades no uso de diversas tecnologias, o estagiário também aperfeiçoou suas habilidades interpessoais ao ser nomeado para participar de reuniões com o cliente. Por fim, destaca-se que as fundamentações teóricas aprendidas no decorrer do curso de Ciência da Computação e a vivência em um projeto real se complementam para a formação de um profissional de excelência.

Palavras-chave: SYDLE ONE. Desenvolvimento *low-code*. Automatização de processo.

ABSTRACT

This document reports the activities done by the author during his internship in software development in a company called SYDLE. Located in Belo Horizonte, it has as its main product the SYDLE ONE: a low-code platform that comprises diverse corporate solutions. In order to be capable of working in the development on SYDLE ONE, the intern took part in a training that presented the platform, as well as introduced the Elasticsearch and BPMN technologies, essential to work on the company‘s projects. After the training, the intern was designated to the project of digital certificate hired by a private company, for 10 months. The project aimed at supplying end-to-end solutions in order to modernize the sale and issue of digital certificates when it comes to the contractor’s service. Thus, the team was responsible for ranging from the maintenance of the e-commerce to the processes to issue and install electronic documents. Besides improving his abilities on the use of diverse technologies, the intern was called for meets with the client several times, which also improved his interpersonal skills. At last, this document emphasizes that the theoretical foundation that was learned during the Computer Science course, in addition to the experience in a real project join together to the background of a great professional.

Keywords: SYDLE ONE. Low-code development. Process automation.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

O curso de Graduação em Ciência da Computação da Universidade Federal de Lavras propõe-se a formar profissionais éticos e interessados na evolução contínua de suas habilidades de criação de sistemas de *software* e *hardware*. Para obter conhecimentos gerais da área, os graduandos devem cursar disciplinas obrigatórias definidas na matriz curricular. Por outro lado, é exigido que o aluno participe de atividades, de escolha pessoal, que se adéquem à sua área de interesse. Dentre essas atividades, o estudante pode realizar um estágio supervisionado e produzir um relatório da experiência para apresentá-lo como trabalho de conclusão de curso (ICET-UFLA, 2021). Dito isso, o presente documento é um relatório de estágio, onde o autor descreve sua vivência como estagiário em desenvolvimento de *software* na empresa SYDLE.

Fundada em 2005 na cidade de Belo Horizonte, a SYDLE oferece um sistema *web*, denominado SYDLE ONE, com diversas soluções corporativas integradas. Essas soluções são completas o suficiente para execução imediata, mas também são passíveis de evolução com desenvolvimento *low-code*. Dentre as soluções, destacam-se o ECM (*Enterprise Content Management*), para gestão de informações empresariais; BPM (*Business Process Management*), para modelagem e automatização de processos; e Analytics, para gerar representações visuais dos dados em tempo real.

O SYDLE ONE é utilizado por diversas empresas espalhadas em mais de 80 países (SYDLE, 2022d). Particularmente, o projeto que colaborei foi contratado por uma empresa privada para melhorar seus serviços de venda e emissão de certificados digitais. Por esse motivo, o escopo do projeto percorria todo o ciclo de vida dos certificados, isto é, éramos responsáveis desde a manutenção do *e-commerce* até a automatização dos processos relacionados à certificação.

As equipes da SYDLE são multidisciplinares e adaptáveis para suprir as necessidades de seus projetos. Dessa forma, a equipe que integrei atendia todos os tipos de demanda, como correções de *bugs* ou evoluções do sistema. Entretanto, para melhorar a qualidade do serviço prestado ao cliente, o time foi dividido em três frentes autossuficientes com propósitos específicos: (1) Estruturante, (2) Evolutiva e (3) Sustentação. A Sustentação, em que fui alocado, encarregava-se de preservar o funcionamento correto do sistema. Para gerenciar nossas demandas, utilizávamos a metodologia Kanban e alguns ritos do Scrum. Ao todo, a equipe contava com quatro desenvolvedores, uma especialista em sucesso do cliente, um analista de qualidade

e uma dona do produto. Em especial, o analista de qualidade e a dona do produto eram funcionários da empresa contratante do projeto.

Antes de entrar no projeto, participei do treinamento de desenvolvimento no SYDLE ONE oferecido pela empresa. Já no projeto, minhas responsabilidades envolviam a manutenção do *e-commerce*, processos e serviços relacionados à certificação digital. Nessas atividades, era utilizado JavaScript, Angular e os conhecimentos de Elasticsearch e BPMN (*Business Process Model and Notation*) adquiridos no treinamento. Após minha alocação no time de Sustentação, atuei, majoritariamente, na correção de *bugs* informados pelos usuários. A partir desse momento, tornou-se comum a minha participação em reuniões virtuais com o cliente para compreensão dos problemas encontrados. Depois de adquirir experiência suficiente, fui escalado para atividades que extrapolavam o desenvolvimento, como o auxílio às equipes de infraestrutura na investigação de problemas de desempenho do sistema e participação em encontros virtuais com membros da área de tecnologia do cliente para instruí-los no uso da plataforma. Por fim, desenvolvi um gerador de grafos de dependências para verificar a precedência dos itens a serem implantados em produção.

Para relatar as principais atividades que desempenhei durante o estágio, o presente trabalho está organizado em quatro capítulos. O Capítulo 1 apresenta a empresa em que o estágio foi realizado e introduz o projeto, a equipe e as atividades que participei. O Capítulo 2 descreve brevemente cada uma das ferramentas que utilizei enquanto estagiário. O Capítulo 3 detalha minuciosamente o treinamento recebido, o projeto, a equipe e as atividades de sustentação do sistema que realizei. O Capítulo 4 descreve os impactos do estágio na minha formação como profissional, destaca a importância da combinação de conhecimentos teóricos com a experiência prática no desenvolvimento de *software* e, por fim, explica duas sugestões de melhoria para recursos relacionados ao SYDLE ONE.

2 CONCEITOS E TECNOLOGIAS

Este capítulo apresenta as principais ferramentas que utilizei durante o estágio. Inicialmente, os processos ágeis Scrum (Seção 2.1) e Kanban (Seção 2.2) são descritos para o contexto de desenvolvimento de *software*. Em seguida, as ferramentas de desenvolvimento *web* são explicadas nesta ordem: HTML (Seção 2.3), JavaScript (Seção 2.4), Angular (Seção 2.5), CSS (Seção 2.6) e Sass (Seção 2.7). Logo após, há uma breve explicação sobre o versionador de códigos Git (Seção 2.8). Posteriormente, aprofundamos sobre o sistema de buscas Elasticsearch (Seção 2.9). Depois, são esclarecidos os símbolos e usos da notação BPMN (Seção 2.10). Na sequência, as principais características das plataformas *low-code* são descritas (Seção 2.11). Por fim, apresenta-se a plataforma *low-code* de soluções corporativas, chamada SYDLE ONE (Seção 3.1.1).

2.1 Scrum

O Scrum é um método ágil proposto para a produção de sistemas. Através de uma série de eventos, artefatos e papéis bem definidos, esse método garante um desenvolvimento flexível e responsivo às mutações de requisitos ocorridas durante a produção (SCHWABER, 1997). Ainda que essa metodologia tenha sido proposta no contexto de desenvolvimento de sistemas, o Scrum pode ser aplicado na fabricação de qualquer tipo de produto (VALENTE, 2020).

Apesar de os times Scrum serem necessariamente pequenos, o conjunto de habilidades dos membros deve ser capaz de suprir todas as exigências do projeto, isto é, o time deve ser autossuficiente para produzir o resultado esperado. A equipe deve ser formada por um Dono do Produto (ou PO, do inglês *Product Owner*), um Scrum Master e desenvolvedores (SCHWABER; SUTHERLAND, 2020).

Segundo (VALENTE, 2020), as atribuições dos papéis exercidos no Scrum são:

- O PO é o representante do cliente no projeto. Ele deve sentir a necessidade do cliente e possuir a ideia clara do produto a ser gerado. Por isso, o PO também é responsável por escrever as histórias de usuário[1] e, consequentemente, deve ter alta disponibilidade para auxiliar o restante do time nas dúvidas que surgirem.

- O Scrum Master é um profissional especialista em Scrum. Sua função é assegurar que todas as regras e rituais do método estão sendo seguidas corretamente. Além de instruir

[1] Histórias de usuário são descrições simples dos requisitos do sistema (VALENTE, 2020).

os demais membros sobre os princípios de Scrum, ele deve atuar como um facilitador dos trabalhos e removedor de impedimentos.

- Por fim, o Desenvolvedor é o especialista técnico que busca e implementa soluções para gerar o produto desejado.

O desenvolvimento no Scrum é feito em pequenos ciclos — chamados *Sprints* — para gerar subprodutos que, quando combinados, formam o produto final. Por isso, esse método é dito iterativo e incremental (VALENTE, 2020).

Além dos *Sprints*, outros eventos realizados no Scrum são o Planejamento do *Sprint*, Reunião Diária (ou *Daily*, em inglês), Revisão e Retrospectiva. Apenas dois desses eventos serão citados no decorrer do presente trabalho: as Reuniões Diárias, nas quais cada membro da equipe relata seu trabalho do dia anterior e seu planejamento para o dia corrente; e a Retrospectiva, uma reunião realizada no último dia do *Sprint* para que o time discuta sobre o ciclo que se encerra e, possivelmente, identifique pontos de melhoria no processo, no relacionamento entre pessoas, ou em quaisquer outros itens importantes para a equipe e o projeto (VALENTE, 2020).

Dentre os artefatos do Scrum, o único mencionado neste documento é o *Backlog* do Produto. Esse instrumento é uma lista de histórias do usuário a serem desenvolvidas. As histórias dessa lista são priorizadas no início de cada *Sprint* pelo PO, conforme as necessidades do cliente (VALENTE, 2020).

2.2 Kanban

Kanban é um processo de produção *just-in-time* que surgiu em indústrias japonesas na década de 1950 (VALENTE, 2020). Por ser genérico o suficiente, pode ser utilizado no gerenciamento de qualquer tipo de produção: de fabricação de carros a desenvolvimento de *software*.

A característica principal desse processo é a utilização de um número restrito de cartões sinalizadores para limitar a quantidade de trabalho em progresso. Para isso, todo novo trabalho deve apropriar-se de um desses cartões e liberá-lo quando for finalizado. Caso não haja cartão disponível, um novo trabalho deve aguardar em uma fila para ser iniciado no momento em que algum cartão for liberado. Portanto, desde que a quantidade de cartões seja definida corretamente, a adoção de Kanban evita a sobrecarga de trabalho. Ressalta-se que no desenvolvimento de *software* os cartões são virtuais e, na verdade, são representados pelos itens de trabalho (ANDERSON, 2010).

Os cartões sinalizadores são afixados no Quadro Kanban. Esse quadro possui colunas para cada uma das etapas que o item percorre para ser produzido. Por isso, o processo de produção deve ser bem definido para que o quadro possa ser elaborado. Da mesma forma, a equipe deve estabelecer o limite de trabalho em progresso para cada etapa. Assim, os cartões são movidos nas colunas para simbolizar a situação de momento do seu trabalho associado. Isso permite que o fluxo de trabalho seja facilmente verificado por qualquer membro da equipe e impede que o trabalho se acumule em etapas específicas (ANDERSON, 2010).

Em desenvolvimento de *software*, a primeira coluna do Quadro Kanban é reservada às histórias do usuário que estão aguardando desenvolvimento, isto é, o *Backlog* do Produto. As colunas seguintes representam as etapas necessárias para que o desenvolvimento seja concluído. Além disso, cada coluna deve ter duas subdivisões: a primeira para alocar os trabalhos em andamento e a segunda para os concluídos. Em outras palavras, uma etapa hipotética denominada "Testes" seria subdividida em "Testando" (em andamento) e "Testado" (concluído) (VALENTE, 2020).

Os precursores da metodologia Kanban em desenvolvimento de *software* argumentam que a redução do trabalho em progresso — e a consequente diminuição de sobrecarga do time — proporciona uma alta frequência de entregas. Por fim, apesar de ser bem caracterizado, Kanban é um sistema totalmente adaptável em relação à papéis e ritos, cabendo a cada equipe definir esses elementos (ANDERSON, 2010).

2.3 HTML

HTML (*HyperText Markup Language*) é uma linguagem de marcação utilizada para definir a estrutura de páginas *web*. Essa linguagem possibilita a criação de documentos com informações e elementos de diferentes níveis de complexidade. De forma resumida, HTML fornece mecanismos para inserir títulos, textos, parágrafos, hiperlinks, tabelas, listas, fotos, vídeos e áudios diretamente nos documentos (W3C, 2016).

Atualmente, as aplicações *web* que utilizam HTML possuem os mais variados propósitos: desde a viabilização de compras on-line até a formação de redes sociais. Contudo, essa linguagem foi criada pelo físico Tim Berners-Lee, em 1991, para facilitar o compartilhamento e acesso a trabalhos científicos a fim de fomentar a colaboração nas pesquisas. Desde 2019,

a padronização do HTML é feita em parceria pelas organizações W3C[2] e WHATWG[3] (W3C, 2019).

A estrutura de um documento HTML é equivalente a uma árvore hierárquica de textos e elementos, sendo os elementos indicados por *tags* de abertura e fechamento. Por sua vez, as *tags* são delimitadas pelos caracteres de menor que (<) e maior que (>). A *tag* raiz dos documentos HTML, por exemplo, é aberta com `<html>` e fechada com `</html>`. Um elemento também pode possuir atributos dentro da *tag* de abertura, que são indicados por nome e valor separados por sinal de igual. Contudo, o valor de um atributo pode ser suprimido se for vazio. Dessa forma, as *tags* com atributos `<a href="https://ufla.br">` e `<details open>` são válidas (WHATWG, 2022).

Os processadores da linguagem, como os navegadores *web*, leem as marcações HTML e transformam-nas em uma árvore DOM[4]. Essa árvore, que reproduz a estrutura do documento em nós, é armazenada em memória e, assim, é possível que *scripts* (geralmente, da linguagem JavaScript) manipulem dinamicamente o conteúdo dos documentos ao alterar os nós do DOM (WHATWG, 2022).

2.4 JavaScript

JavaScript é uma linguagem de programação interpretada, de alto nível e baseada em protótipos. Além disso, é multiparadigma pois oferece funcionalidades para programação orientada a objeto, funcional e imperativa (MDN WEB DOCS, 2021c).

A linguagem foi criada em 1995 por Brendan Eich para rodar no Netscape Navigator[5] e, então, oferecer maior dinamismo e interatividade em páginas *web* (SEVERANCE, 2012). Após o lançamento do JavaScript, outros navegadores implementaram linguagens semelhantes com o mesmo propósito. Dessa forma, no final de 1996, o JavaScript passou a ser normalizado pela Ecma International[6] para torná-lo um padrão do setor (MDN WEB DOCS, 2021b). Essa normalização, que recebeu o nome de ECMAScript, foi lançada pela primeira vez em 1997

[2] A W3C (*World Wide Web Consortium*) é uma organização fundada, em 1994, por Tim Berners-Lee para padronizar os recursos *web*, incluindo o HTML

[3] A WHATWG (*Web HyperText Applications Technology Working Group*) é uma organização fundada, em 2004, por empresas ligadas a fabricação de navegadores com o objetivo de acelerar a especificação do HTML e tecnologias relacionadas.

[4] O DOM (*Document Object Model*) é uma especificação da W3C para representar e manipular o conteúdo de documentos de linguagem de marcação.

[5] Netscape Navigator foi um navegador *web* da empresa Netscape Communications Corporation.

[6] Ecma International é uma associação europeia de normalização de sistemas de informação.

garantindo, desde então, estabilidade nas implementações da linguagem (MDN WEB DOCS, 2021d) .

Particularmente, JavaScript é uma linguagem de programação que não possui mecanismos entrada e saída. Os *scripts* dessa linguagem devem ser executados em um ambiente hospedeiro que, por sua vez, oferece a comunicação com o mundo real. Apesar de o hospedeiro inicial — e mais comum até os dias atuais — ser o navegador *web*, é possível encontrar interpretadores da linguagem nos mais diversos ambientes, como servidores e bancos de dados não relacionais (MDN WEB DOCS, 2021d).

A linguagem, que possui tipagem fraca e dinâmica, dispõe de seis tipos primitivos: (1) número, (2) *string*, (3) booleano, (4) objeto, (5) nulo e (6) indefinido (MDN WEB DOCS, 2021d). Por fim, destaca-se que em JavaScript as funções são objetos, já que a linguagem implementa o conceito de funções de primeira classe (MDN WEB DOCS, 2021c). Por consequência, as funções são tratadas como variáveis comuns, podendo ser retornadas por outras funções ou serem passadas como parâmetro.

2.5 Angular

Angular é uma plataforma e *framework* para produção de aplicações de página única. Esse *framework* permite criar e manipular elementos de interface em aplicações *web* utilizando TypeScript[7]. Algumas bibliotecas acompanham o Angular para suprir necessidades frequentes na construção das aplicações, como gerenciamento de formulários e comunicação cliente-servidor (ANGULAR, 2022c).

O *framework* é baseado em componentes de interface dinâmicos e reaproveitáveis. Um componente é definido por uma classe TypeScript e duas propriedades informadas obrigatoriamente no arquivo da classe: seletor e *template*. A classe do componente é como uma classe qualquer, suportando todo tipo de lógica que a linguagem permite. O seletor determina o nome do elemento HTML utilizado para inserir o componente em uma página. Assim, um componente com o seletor `meu-componente` é inserido em uma página HTML através da marcação `<meu-componente></meu-componente>`. Já a propriedade *template* define o conteúdo HTML do componente. Em outras palavras, o *template* informa como o componente será renderizado (ANGULAR, 2022c).

[7] TypeScript é uma linguagem de programação que implementa JavaScript com tipagem forte e todos os recursos de orientação a objeto.

Ainda no *template*, podemos inserir valores dinâmicos através da interpolação de dados implementada pelo Angular. Para isso, sempre que o estado do componente é alterado, o *framework* se encarrega de alterar a árvore DOM (Seção 2.3) do documento HTML. O caminho inverso é igualmente possível, isto é, alterar dinamicamente o estado do componente através da manipulação da interface. Essa técnica, chamada de *two-way data biding*, garante que dados permaneçam sincronizados na interface e no objeto TypeScript (ANGULAR, 2022c).

Outro ponto de destaque do Angular são as diretivas: um conjunto de comandos para adicionar comportamentos dinâmicos aos *templates*. As diretivas permitem, por exemplo, condicionar os estilos e a exibição de elementos de acordo com algum estado do componente (ANGULAR, 2022a).

Para aumentar a testabilidade e flexibilidade do código, Angular implementa o padrão de projeto Injeção de Dependência. Nesse padrão, as dependências de uma classe são criadas externamente e inseridas quando forem necessárias. No caso do Angular, as dependências a serem injetadas são informadas no construtor da classe e o próprio *framework* é responsável por criá-las (ANGULAR, 2022b).

A plataforma ainda oferece ferramentas para facilitar seu uso em projetos. A título de exemplo, o Angular CLI é uma interface de linha de comando para gerenciar todas as etapas de desenvolvimento e execução de aplicações Angular. Os comandos disponíveis possibilitam, especialmente, inicializar projetos, rodar testes unitários e construir a aplicação para executá-la (ANGULAR, 2016).

2.6 CSS

CSS (*Cascading Style Sheets*) é uma linguagem declarativa utilizada para definir estilos de aparência para documentos HTML. Em 1997, a W3C lançou o HTML 3.2 que permitia a alteração de atributos visuais dos documentos, como fontes e cores. Contudo, essas alterações eram feitas através de *tags* HTML, impossibilitando o reaproveitamento de estilos. Então, a própria W3C criou a linguagem CSS para que as configurações de aparência se tornassem independentes do documento HTML e pudessem ser utilizadas em diferentes páginas (W3SCHOOLS, 2007).

Os códigos CSS podem ser inseridos diretamente no HTML dentro de um elemento `<style>`. No entanto, para haver o reaproveitamento de estilos em diferentes páginas, deve-se criar arquivos CSS externos e referenciá-los nos documentos HTML através da *tag* `<link>`. A

terceira forma — menos recomendada — de estilizar HTML é inserir propriedades CSS no atributo `style` do elemento que será modificado (MDN WEB DOCS, 2021a).

CSS permite manipular a aparência de documentos através de um conjunto de regras de estilo definidas por uma sintaxe simples. As regras são iniciadas com o seletor de um ou mais elementos que receberão o estilo. Esse seletor pode ser, por exemplo, o tipo, classe, identificador ou atributo de um elemento. Após o seletor, declara-se uma ou mais definições de estilo separadas por ponto e vírgula. Cada declaração deve informar a propriedade a ser modificada e seu valor desejado. Exemplificando, caso o texto dos parágrafos de um documento HTML devam apresentar-se centralizados e na cor vermelha, a regra CSS definida seria `p { text-align: center; color: red; }` (W3SCHOOLS, 2017).

Os navegadores *web* são os responsáveis pela renderização dos estilos definidos em CSS. Resumidamente, após converter o HTML em árvore DOM (Seção 2.3), o navegador analisa o CSS e gera a árvore de renderização ao inserir as regras nos nós pertinentes da árvore. Por fim, a visualização estilizada é gerada e exibida ao usuário (MDN WEB DOCS, 2020).

2.7 Sass

Sass (*Syntactically Awesome Style Sheets*) é uma linguagem de declaração de estilos que, quando compilada, gera códigos CSS (Seção 2.6) equivalentes. A linguagem suporta a criação de variáveis, regras aninhadas, heranças e outras funcionalidades para facilitar a escrita e aproveitamento de estilos. Sass é considerado uma extensão do CSS por possuir compatibilidade total com sua sintaxe (SASS, 2013a). Apesar de ser originalmente escrita em Dart[8], Sass é distribuída em JavaScript puro em um pacote no npm[9] (SASS, 2018).

As variáveis em Sass são definidas com o símbolo cifrão (`$`) e podem armazenar, além dos valores aceitos em CSS, valores booleanos, mapas e referências para funções (SASS, 2019b). A declaração de uma variável é semelhante a de uma propriedade CSS: inicia-se com o nome da variável, seguida de dois-pontos e, por fim, seu valor. A variável pode ser referenciada em diferentes partes do arquivo, facilitando o reúso e diminuindo a repetição de código. Ao compilar o código Sass, todas as referências para variáveis são substituídas pelo seu valor e, assim, é gerado um CSS válido (SASS, 2013b). A Figura 2.1 mostra um exemplo de compilação de variáveis.

[8] Dart é uma linguagem de *script* para aplicações *web* desenvolvida pela Google.
[9] O npm é um gerenciador de pacotes para o interpretador de JavaScript chamado Node.js.

Figura 2.1 – Exemplo de compilação de código Sass com uso de variável

Código SASS

```
$variavel-de-cor: #1D4F91;

h1 {
  color: $variavel-de-cor;
}

.box {
  background-color: $variavel-de-cor;
}
```

Código CSS compilado

```
h1 {
  color: #1D4F91;
}

.box {
  background-color: #1D4F91;
}
```

Legenda: No código SASS, a declaração da variável e seus usos estão em vermelho. Já no código CSS, a substituição da variável está escrito em vermelho.

Fonte: Do autor (2022).

A fim de criar uma hierarquia semelhante àquela dos elementos HTML, Sass possibilita o aninhamento de seletores para definição de estilos. Além disso, é possível criar módulos de estilos que podem ser incluídos em diferentes arquivos para evitar a repetição de código (SASS, 2013b). A Figura 2.2 exemplifica, simultaneamente, o aninhamento de seletores e o uso de módulos.

Figura 2.2 – Exemplo de compilação de código Sass com aninhamento de seletores e uso de módulo

Módulo SASS

```
// _baseStyle.scss
body {
  margin: 0;
}
```

Código SASS

```
@use 'baseStyle'

.box {
  button {
    background: limegreen;
  }

  a {
    color: limegreen;
  }
}
```

Código CSS compilado

```
body {
  margin: 0;
}

.box button {
  background-color: limegreen;
}

.box a {
  color: limegreen;
}
```

Legenda: No código SASS, o uso de módulo está em vermelho e a implementação de grupos aninhados está em verde. No código CSS, as cores vermelha e verde representam a compilação dos trechos de mesma cor no código SASS.

Fonte: Do autor (2022).

Sass implementa o conceito de herança para o contexto de estilos. Isso significa ser possível criar grupos de declarações CSS extensíveis para que outros grupos herdem suas declarações (SASS, 2013b). O uso de herança de estilos pode ser visto na Figura 2.3.

Os recursos do Sass também permitem a criação de funções, uso de estruturas condicionais e de repetição (SASS, 2013a). Ainda é possível executar cálculos matemáticos utilizando os operadores `+` (adição), `-` (subtração), `*` (multiplicação) e `%` (resto). Em adição, Sass dis-

Figura 2.3 – Exemplo de compilação de código Sass com herança

Código SASS

```
// Grupo de estilos extensível
%default-button {
  padding: 5px;
  color: white;
}

// Grupo que estende %default-button
.confirmation-button {
  @extends %default-button;
  background-color: limegreen;
}

// Grupo que estende %default-button
.cancel-button {
  @extends %default-button;
  background-color: red;
}
```

Código CSS compilado

```
.confirmation-button, .cancel-button {
  padding: 5px;
  color: white;
}

.confirmation-button {
  background-color: limegreen;
}

.cancel-button {
  background-color: red;
}
```

Legenda: No código SASS, a declaração do grupo de estilos extensível está em vermelho e o comando para estendê-lo está em verde.

Fonte: Do autor (2022).

ponibiliza um conjunto de módulos nativos, incluindo um para operações matematicas. Esse módulo contém, entre outros recursos, funções para cálcular divisão, raiz e para gerar números aleatórios (SASS, 2019a).

2.8 Git

Git é um sistema distribuído que realiza o versionamento de códigos (GIT, 2012b). Apesar de permitir que repositórios remotos sejam clonados localmente, o versionador possui recursos para garantir que um arquivo não seja sobrescrito ao aplicar atualizações concorrentes no repositório central (GIT, 2012a).

Nativamente, Git disponibiliza ferramentas de linha de comando para execução de suas funcionalidades. Para exemplificar, a criação de um novo repositório é feita através do comando `git init`. Também, existem programas de terceiros para execução de comandos Git por interface gráfica. Entretanto, apenas a linha de comando garante compatibilidade total com os recursos do Git (CHACON; STRAUB, 2014).

Como outros versionadores, Git implementa o conceito de *branches* (ramificações, em tradução livre). Dessa forma, a partir de um estado do repositório, o usuário pode criar vertentes (ou ramos) de desenvolvimento distintas. O comando `git branch nome-da-ramificacao` é usado para criar um *branch*. Também, é possível alternar entre essas ramificações por meio do comando `git checkout nome-da-ramificacao` (CHACON; STRAUB, 2014).

Para os casos em que é necessário unir *branches* distintos, Git fornece o comando `merge` (CHACON; STRAUB, 2014). Em uma situação hipotética, esse comando seria utili-

zado quando vários desenvolvedores implementam funcionalidades em *branches* distintas e, ao concluir seus trabalhos, precisam aplicar suas implementações em um *branch* principal.

O controle de alterações em arquivos no repositório são feitos, da maneira mais básica, pelos comandos `add`, `commit`, `pull` e `push`. Os comandos `add` e `commit` são utilizados para atualizar os itens do repositório local. Por outro lado, `pull` busca as modificações mais recentes do repositório remoto e aplica no local, enquanto `push` envia as modificações do local para o remoto (VALENTE, 2020). Existem outros comandos essenciais para a utilização de Git, mas não serão detalhados por não serem citados no presente trabalho.

2.9 Elasticsearch

Elasticsearch é uma ferramenta para busca e análise de dados. Qualquer tipo de dado, estruturado ou não, pode ser armazenado e indexado pela ferramenta: de textos e números a arquivos e dados geográficos (ELASTIC, 2019d). Ainda são fornecidas APIs REST para executar consultas e manipulação dos dados indexados (ELASTIC, 2019c).

Ao serem inseridos no Elasticsearch, os dados são serializados em estruturas JSON[10] chamadas de documentos. Dessa forma, as informações dos documentos são armazenadas em uma estrutura de chave e valor que serão indexadas (ELASTIC, 2019a).

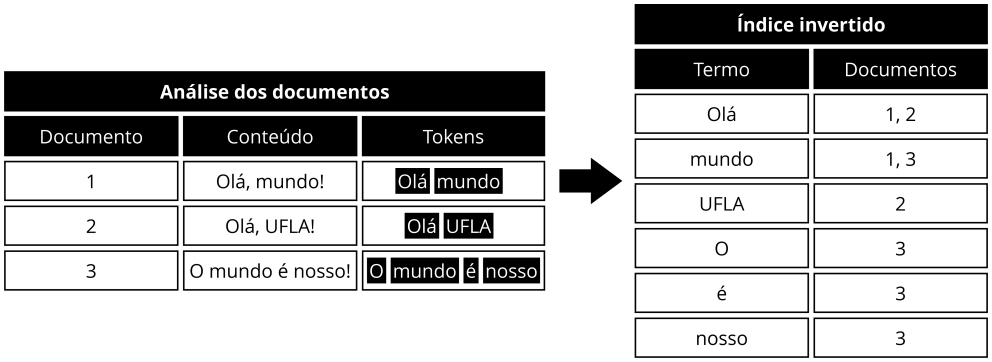
A indexação dos documentos é feita de diferentes maneiras considerando o tipo de cada um de seus campos. Para os campos do tipo texto, por exemplo, os documentos são indexados utilizando a estratégia de índice invertido. Isto é, como exemplificado na Figura 2.4, todos os termos presentes em campos de tipo texto dos documentos são mapeados e, por outro lado, cada termo possui uma lista de referências para os documentos onde ele ocorre. Outros tipos de dados são indexados utilizando estruturas otimizadas para suas características. É importante saber que a identificação do tipo de dado dos campos é feita, por padrão, de forma automática pelo Elasticsearch, reduzindo o trabalho para inserir documentos nos índices (ELASTIC, 2019a).

Elasticsearch ainda disponibiliza algumas APIs para realizar a indexação, busca e análise dos dados. Algumas linguagens, incluindo JavaScript, possuem bibliotecas que facilitam o uso dessas APIs em aplicações com papel de cliente. Além de ser possível realizar buscas e agregações nos dados utilizando consultas no estilo SQL[11], Elasticsearch também implementa uma linguagem de consulta em forma de JSON. As buscas e agregações podem ser feitas ao

[10] JSON (JavaScript Object Notation) é um formato de estruturação de dados para serem transmitidos entre sistemas.

[11] SQL (*Standard Query Language*) é uma linguagem de consulta para bancos de dados relacionais.

Figura 2.4 – Exemplo básico de construção de índice invertido



Fonte: Do autor (2022).

mesmo tempo em uma única requisição, promovendo um alto desempenho para análise de dados em tempo real (ELASTIC, 2019b).

2.10 BPMN

BPMN (*Business Process Model & Notation*) é uma especificação de notação gráfica para modelagem de processos de negócio. A notação, que se assemelha a um fluxograma, consegue representar, visualmente, processos complexos utilizando uma simbologia padronizada e compreensível para os envolvidos na implementação desses processos (OMG, 2018).

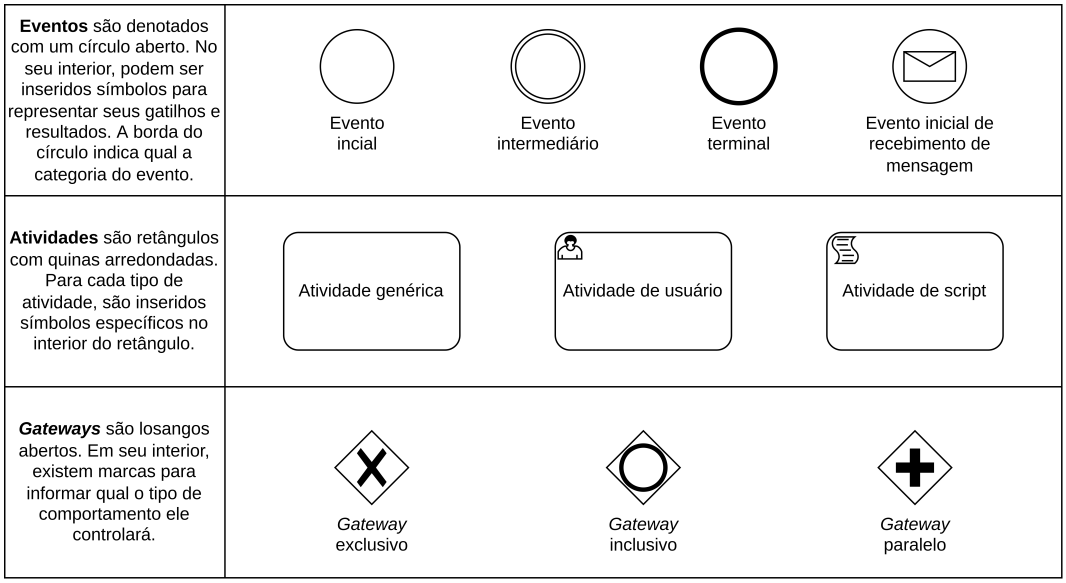
A especificação contém um mapeamento da notação gráfica para uma linguagem de execução processos. Esse mapeamento visa uniformizar o entendimento dos processos nas diferentes áreas interessadas. Dessa forma, os profissionais de negócios projetam processos utilizando diagramas BPMN que, posteriormente, serão facilmente estruturados em linguagem de execução por programadores encarregados de desenvolver os sistemas computacionais relacionados (OMG, 2013). Entretanto, levando em consideração o escopo do presente trabalho, exploraremos apenas os diagramas BPMN.

Para a elaboração de diagramas de processos, o BPMN define elementos divididos em cinco categorias: (1) objetos de fluxo, (2) dados, (3) objetos de conexão, (4) raias e (5) artefatos (OMG, 2013). Mais uma vez, considerando a abordagem do presente trabalho, não será necessário discorrer sobre os elementos de dados.

Os objetos de fluxo são usados para definir o comportamento do processo, por isso são os principais elementos de um diagrama. Existem três tipos de objetos de fluxo com propósitos específicos: (1) eventos, para representar acontecimentos que alteram o fluxo do processo e são ativados a partir de gatilhos; (2) atividades, para representar os trabalhos a serem executados

no decorrer do processo e; (3) *gateways*, para controlar a divergência e convergência no fluxo. Além disso, para cada um desses tipos de objeto, existem elementos para contextos particulares. A título de exemplo, existem eventos iniciais, intermediários e finais; as atividades podem ser de usuário, de serviço ou *script*; e há *gateways* exclusivos, inclusivos e paralelos (OMG, 2013). A Figura 2.5 exibe a notação de cada um dos tipos de objetos de fluxo.

Figura 2.5 – Notação dos objetos de fluxo do BPMN 2.0

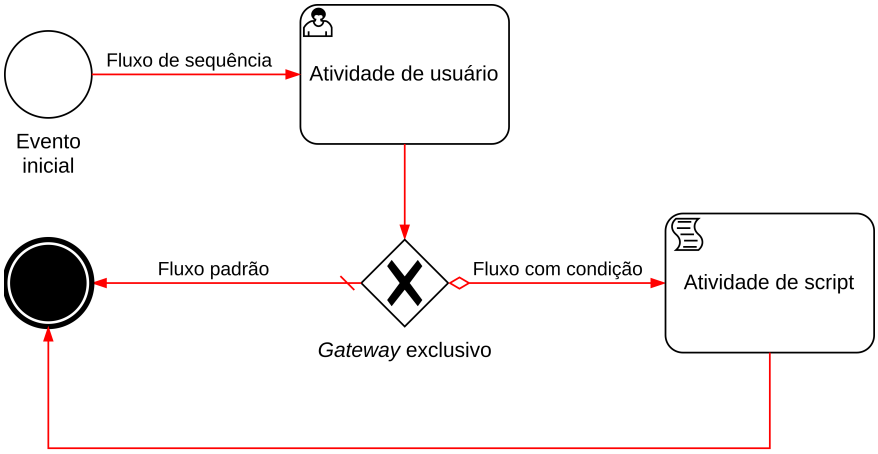


Fonte: Do autor (2022).

Os objetos de conexão são separados em quatro categorias: (1) fluxo de sequência, (2) fluxo de mensagem, (3) associação e (4) associação de dados. Em especial, o fluxo de sequência é utilizado para exibir a ordem de precedência das atividades do processo. Como pode ser visto na Figura 2.6, objetos desse tipo são denotados por setas com a ponta preenchida, sendo utilizados para conectar objetos de fluxo. Ao serem combinados com *gateways*, os fluxos de sequência podem possuir condições para serem percorridos ou serem definidos como o caminho padrão caso nenhum outro fluxo condicional seja ativado. Os fluxos de sequência com condição são denotados com um pequeno losango no início da seta. De forma análoga, os fluxos definidos como padrão possuem um traço transversal também no início da seta (OMG, 2013).

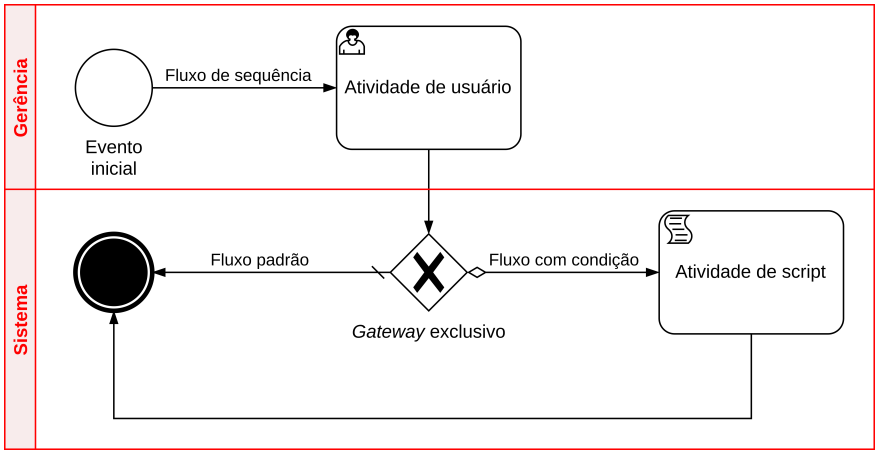
As raias informam, graficamente, os participantes do processo responsáveis pelas atividades contidas em seu escopo. Em outras palavras, todas as atividades incluídas em uma raia só poderão ser executadas pelos participantes definidos por ela (OMG, 2013). A notação da raia pode ser vista na Figura 2.7.

Figura 2.6 – Exemplo de uso de fluxos de sequência



Legenda: Em vermelho estão os fluxos de sequência.
Fonte: Do autor (2022).

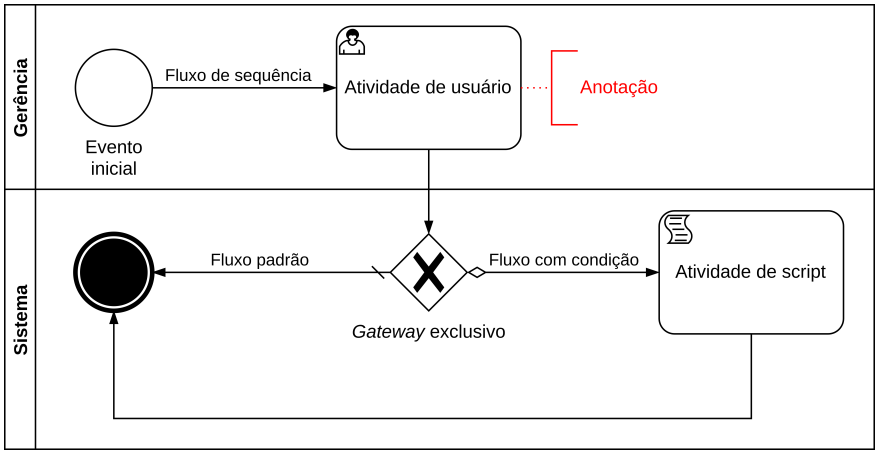
Figura 2.7 – Uso de raias em diagramas de processos



Legenda: As raias, assinaladas em vermelho, indicam que apenas usuários pertencentes à gerência são aptos a atender a "Atividade de usuário" e o restante do processo é executado pelo sistema.
Fonte: Do autor (2022).

Por fim, os artefatos são utilizados para aumentar a riqueza de informações nos processos e não interferem em sua execução. Existem dois artefatos padronizados: (1) anotação de texto, utilizado para armazenar informações adicionais de algum elemento do diagrama e (2) grupo, utilizado para agrupar visualmente elementos de uma mesma categoria. Apesar de existirem dois artefatos especificados, BPMN permite que novos artefatos sejam criados conforme a necessidade (OMG, 2013). A Figura 2.8 exemplifica o uso de uma anotação de texto.

Figura 2.8 – Exemplo de uso de anotação de texto



Fonte: Do autor (2022).

2.11 Plataforma *low-code*

As plataformas *low-code*, em geral, permitem a criação de sistemas utilizando o mínimo de escrita códigos em linguagens de programação tradicionais. Para isso, essas plataformas permitem que aplicações sejam criadas mais rapidamente através da interface gráfica do usuário. (BOCK; FRANK, 2021).

Atualmente, o termo "*low-code*" não possui uma definição formal. Por consequência, existe uma grande variação dos propósitos e características das plataformas que se autointitulam "*low-code*". Contudo, segundo (BOCK; FRANK, 2021), as plataformas desse tipo incluem, em único ambiente, as ferramentas e componentes necessários para a criação de algum tipo de sistema. O tipo do sistema dependerá do propósito da plataforma.

É comum que em plataformas *low-code* existam interfaces gráficas para projetar o sistema, componentes para modelar estruturas de dados e configurações para definição de papéis e permissões de usuários. Em algumas plataformas é possível criar componentes utilizando linguagens de programação já conhecidas, como JavaScript (Seção 2.4), ou, até mesmo, modelar processos utilizando BPMN (Seção 2.10) (BOCK; FRANK, 2021).

3 O ESTÁGIO

Este capítulo apresenta as principais atividades que desempenhei durante o estágio na empresa SYDLE. Para começar, descrevo a organização, produtos e serviços da empresa SYDLE (Seção 3.1). Na sequência, relato minha participação no treinamento oferecido aos programadores recém-contratados pela empresa (Seção 3.2). Em seguida, após a contextualização da área de negócio do cliente, é descrito o projeto que integrei por 10 meses (Seção 3.3). Depois de explicar a divisão da equipe nos *squads* Estruturante, Evolutivo e Sustentação (Seção 3.4), caracterizo minuciosamente o processo de desenvolvimento no *squad* Sustentação (Seção 3.5). Por fim, descrevo as principais tarefas que executei enquanto integrante do *squad* Sustentação (Seção 3.6).

3.1 A empresa

A SYDLE é uma empresa brasileira do ramo de tecnologia que foi fundada em 2005. Além da sede localizada em Belo Horizonte (MG), existem outras três unidades nas cidades de Viçosa (MG), São Paulo (SP) e Rio de Janeiro (RJ). Diferentemente da sede, as demais unidades não possuem escritório físico e, portanto, todos seus colaboradores atuam em *home office*.

Para a localidade que possui escritório, a SYDLE adota o regime de trabalho híbrido: os colaboradores decidem se trabalharão presencialmente ou de forma remota. Em vista disso, o estágio descrito neste documento foi realizado nessa unidade com atuação remota.

Atualmente, a SYDLE é uma empresa de produto e serviço. Seu produto principal é o SYDLE ONE: uma plataforma *web* voltada para a criação, personalização e utilização de soluções corporativas. Os demais produtos são implementações de soluções completas e genéricas para uso na plataforma. Já o serviço prestado pela SYDLE compreende à implementação de recursos no SYDLE ONE que atendam às necessidades específicas do cliente. No momento da escrita deste trabalho, a SYDLE possui clientes espalhados em mais de 80 países.

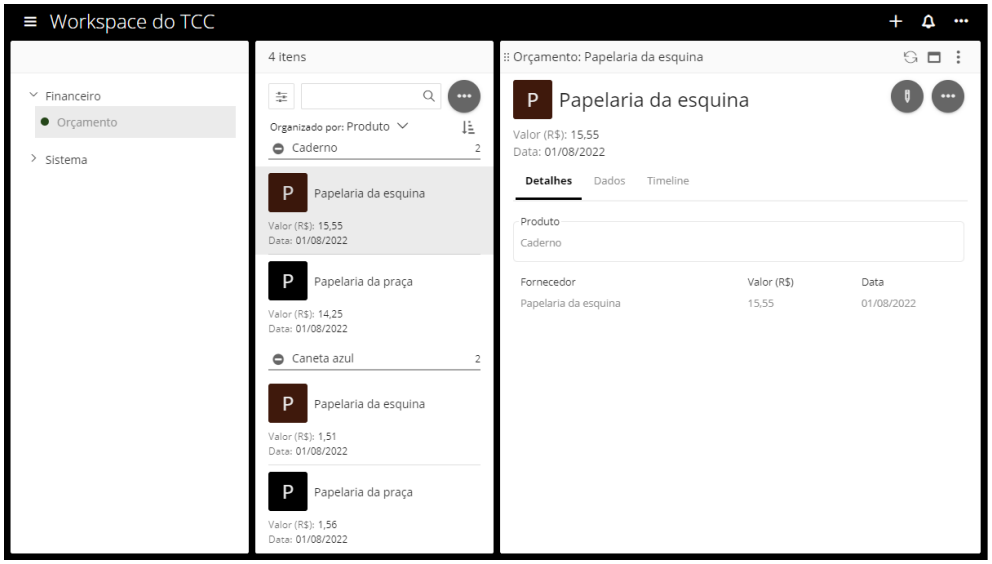
De forma geral, os desenvolvedores de *software* da SYDLE se dividem em quatro tipos de equipes: (1) de produto, responsáveis pelas evoluções do SYDLE ONE; (2) de SYBOX, que criam as soluções genéricas para uso no SYDLE ONE; (3) de projeto, que desenvolvem soluções específicas demandadas por clientes e (4) de *Growth*, que elaboram os portais e serviços que promovem os produtos da SYDLE. O estágio descrito nesse trabalho, relata o cotidiano de uma equipe de projeto.

Todas as equipes são inteiramente livres para seguir as metodologias de gestão que beneficiem a produtividade e o bem-estar dos colaboradores. Até dentro de uma mesma equipe, podem haver diferentes frentes que trabalham com metodologias distintas. Essa liberdade também se estende aos funcionários, pois a empresa utiliza o modelo de horário de trabalho flexível, onde cada colaborador se autogerencia em concordância com sua equipe.

3.1.1 SYDLE ONE

SYDLE ONE é uma plataforma digital, criada e mantida pela empresa SYDLE, com diversas soluções corporativas integradas, como automatização de processos, gestão de documentos, relacionamento com clientes e visualização de dados em tempo real (SYDLE, 2022h). Apesar de possuir soluções prontas, a plataforma também permite a criação de novas funcionalidades com desenvolvimento *low-code*. As soluções de ECM (*Enterprise Content Management*), BPM (*Business Process Management*) e Analytics constituem a base arquitetural do SYDLE ONE. A interface da plataforma pode ser vista na Figura 3.1

Figura 3.1 – Interface do SYDLE ONE



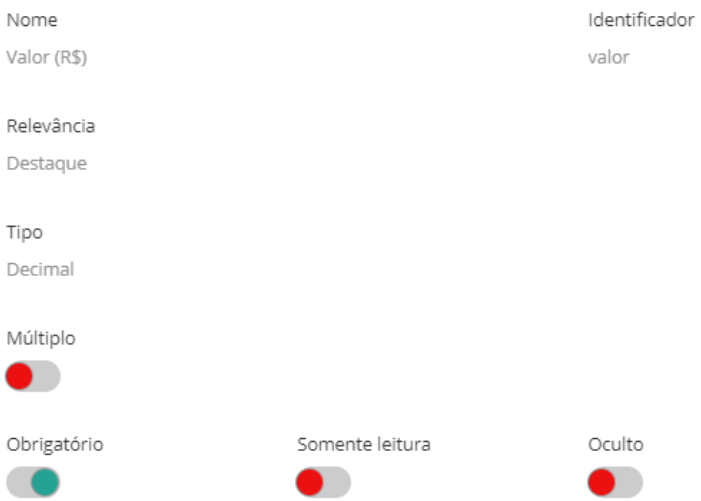
Fonte: Do autor (2022).

Para o ECM, o SYDLE ONE permite a criação de classes de dados e a organização delas em pacotes. Apesar da nomenclatura semelhante, a plataforma não é orientada a objetos de fato, mas segue alguns dos conceitos desse paradigma. Em resumo, para modelar uma classe deve-se definir seu pacote, identificador, campos de dados e métodos em JavaScript. Após a modelagem, o usuário consegue criar objetos das classes e acessá-los pela interface da plataforma ou pelas APIs *web* do SYDLE ONE. As APIs serão descritas ainda nessa seção.

Os itens criados no SYDLE ONE são indexados utilizando Elasticsearch. Por isso, ao criar uma nova classe, são gerados alguns campos para armazenar o índice dos objetos no Elasticsearch e algumas informações de criação e alteração. Também existem métodos criados por padrão que podem ser customizados para, em geral, editar objetos, obtê-los diretamente pelo seu índice no Elasticsearch, buscá-los utilizando a linguagem de consulta do Elasticsearch ou controlar dinamicamente os formulários de criação e edição dos objetos. É importante destacar que a plataforma gera automaticamente os formulários de manipulação de objetos, bastando ao desenvolvedor definir a estrutura da classe e seus comportamentos.

O exemplo visto no Apêndice A, mostra a modelagem da classe "Orçamento", pertencente ao pacote "Financeiro", contendo os campos "Produto", "Fornecedor", "Valor" e "Data". A classe também possui um método "Obter melhor valor" que, por suposição, recebe uma lista de objetos de "Orçamento" e retorna aquele com o menor valor. A Figura 3.2 mostra que os campos possuem metadados para definir, em especial, seu tipo, identificador e obrigatoriedade. É importante saber que o tipo de um campo pode ser — além de números, textos, datas e outros tipos básicos — referências para classes. Quando isso acontece, o campo só poderá ser preenchido com objetos da classe referenciada.

Figura 3.2 – Alguns metadados de campos



Fonte: Do autor (2022).

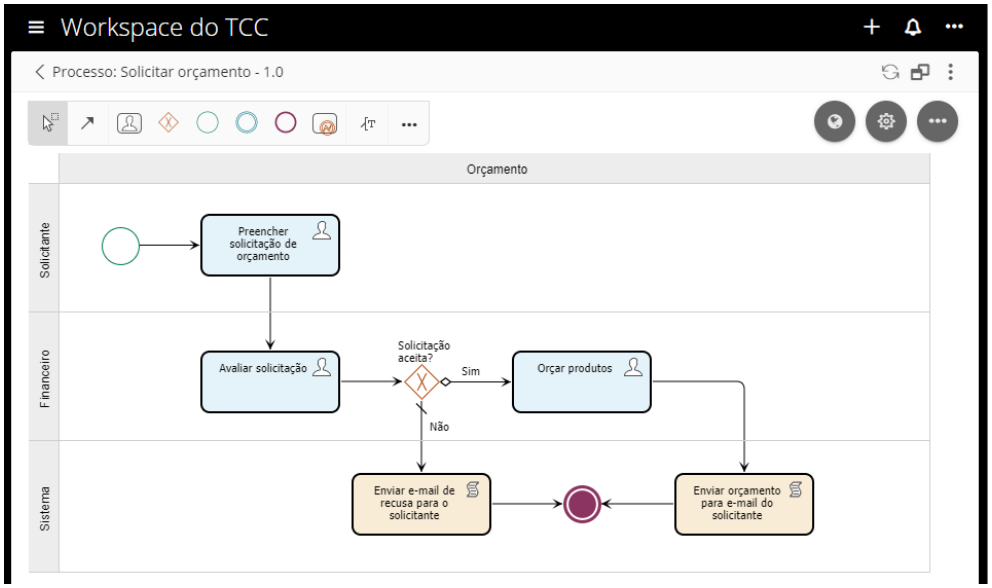
Dado que pacotes, classes e métodos possuem identificadores, a plataforma consegue interpretar chamadas de métodos utilizando uma combinação dessas informações. Voltando ao exemplo da classe "Orçamento", do pacote "Financeiro" e o método "Obter melhor valor", caso

os identificadores sejam, respectivamente, `budget`, `financial` e `getBestPrice`, é possível invocar esse método pelo comando `financial.budget.getBestPrice(listOfBudgets)` em *scripts*.

O SYDLE ONE oferece controle de acesso a qualquer recurso criado na plataforma. Essa funcionalidade é útil para limitar, para determinados usuários ou grupos de usuários[1], o acesso a pacotes, classes, objetos e, até mesmo, métodos. Do mesmo modo, pode-se configurar a geração automática de APIs *web* com restrições de acesso. Como as APIs são usadas para expor métodos de classes, o uso do SYDLE ONE como *back-end* de outras aplicações torna-se viável.

A solução de BPM da plataforma permite modelar processos corporativos utilizando um editor com notação BPMN (Seção 2.10), automatizá-los com JavaScript e gerenciá-los através de gráficos ou visualização do fluxo (SYDLE, 2022c). Para cada raia do processo, pode-se definir perfis de usuários para limitar as pessoas aptas a atenderem as atividades. Esses perfis são associados a usuários ou grupos de usuários. O editor com notação BPMN pode ser visto na Figura 3.3. O SYDLE ONE estende os elementos do BPMN, associando classes de dados a eles. Isso significa que os formulários de preenchimento de dados serão gerados automaticamente nas atividades de usuário (SYDLE, 2022c).

Figura 3.3 – Modelagem de processo “Solicitação de orçamento” no editor de BPMN do SYDLE ONE



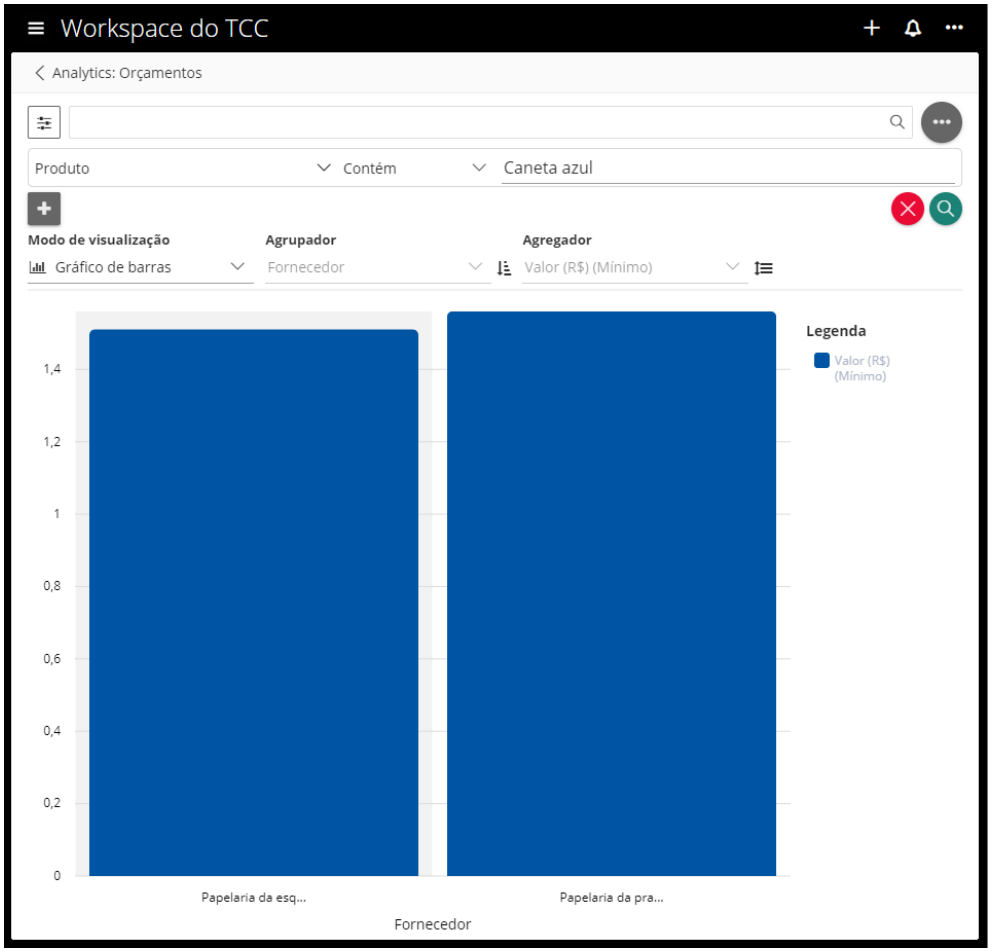
Fonte: Do autor (2022).

Por fim, os Analytics possibilitam uma gestão à vista e em tempo real dos dados armazenados no SYDLE ONE. A partir dos objetos de uma classe pré-definida, são gerados gráficos,

[1] A plataforma possui classes nativas para armazenar os usuários de acesso e os grupos aos quais eles pertencem.

tabelas, *pipelines* e outras visualizações de dados (SYDLE, 2022a). Um exemplo de Analytics é mostrado na Figura 3.4.

Figura 3.4 – Gráfico de orçamentos gerado pelo Analytics



Fonte: Do autor (2022).

3.1.2 SYBOX

SYBOX é o nome dado às soluções desenvolvidas no SYDLE ONE disponibilizadas para utilização imediata ou como modelo. Apesar de estarem prontas, as soluções dos SYBOX podem ser evoluídas para se adequarem a eventuais necessidades do projeto em que serão utilizadas (SYDLE, 2022g).

Entre os SYBOX disponibilizados, destacam-se CRM (*Customer Relationship Management*), Billing e Service Desk. O CRM fornece recursos para gerenciar vendas, clientes, planos de marketing, redes sociais, além de possibilitar a configuração de *chatbots* (SYDLE, 2022e). Já o Billing, é uma solução para gerenciar o clico de vida completo de vendas e disponibiliza integrações com diversos meios de pagamento para realizar cobranças de produtos, serviços

e assinaturas (SYDLE, 2022b). Por último, o Service Desk é uma plataforma para gestão de demandas que permite, por exemplo, a disponibilização de conteúdos de suporte ao usuário, implementar serviço de *tickets* ou a integração com CRM para melhorar a experiência de venda e pós venda do cliente (SYDLE, 2022f).

As soluções genéricas são criadas por times dedicados aos SYBOX. Contudo, em times de projetos para clientes, são gerados SYBOX com objetos, classes, processos ou quaisquer outras entidades criadas e alteradas no ambiente de desenvolvimento para importá-los nos ambientes de homologação e produção. Os SYBOX são importados no SYDLE ONE através de arquivos criptografados com as informações dos itens de interesse. Para gerar um arquivo desse tipo, basta criar um objeto da classe SYBOX contendo uma lista dos artefatos a serem exportados. Ao final da criação do objeto, o SYDLE ONE gera automaticamente o arquivo de SYBOX contendo uma cópia exata dos itens selecionados. Quando o SYBOX é importado no ambiente de destino, os itens já existentes são sobrescritos e os novos são criados.

3.2 Treinamento

Antes de ser alocada em algum projeto, toda pessoa recém-contratada pela SYDLE deve participar de uma série de treinamentos promovidos pela empresa. Desde o primeiro dia, os ingressantes participam de reuniões para instrução das ferramentas utilizadas em sua área de atuação. Dessa forma, é oferecido um treinamento especial para os novos programadores visando capacitá-los para o desenvolvimento de soluções no SYDLE ONE (Seção 3.1.1). Então, na primeira semana iniciei minha capacitação.

O treinamento do SYDLE ONE ocorreu através de encontros e debates virtuais — via Google Meet[2] e Discord[3] — realizados no horário regular de expediente. No primeiro encontro, um instrutor nos apresentou a interface da ferramenta e alguns de seus conceitos fundamentais, como pacotes, classes, processos e grupos de usuários. Também, foi apresentada a documentação do SYDLE ONE, que foi utilizada no decorrer do treinamento.

Após o encontro introdutório, fui orientado a realizar uma atividade prática: modelar um processo corporativo de solicitação de compras. O fluxo do processo deveria seguir um roteiro definido por um conjunto de requisitos. Apesar de a documentação do SYDLE ONE dispor

[2] https://meet.google.com
[3] https://discord.com

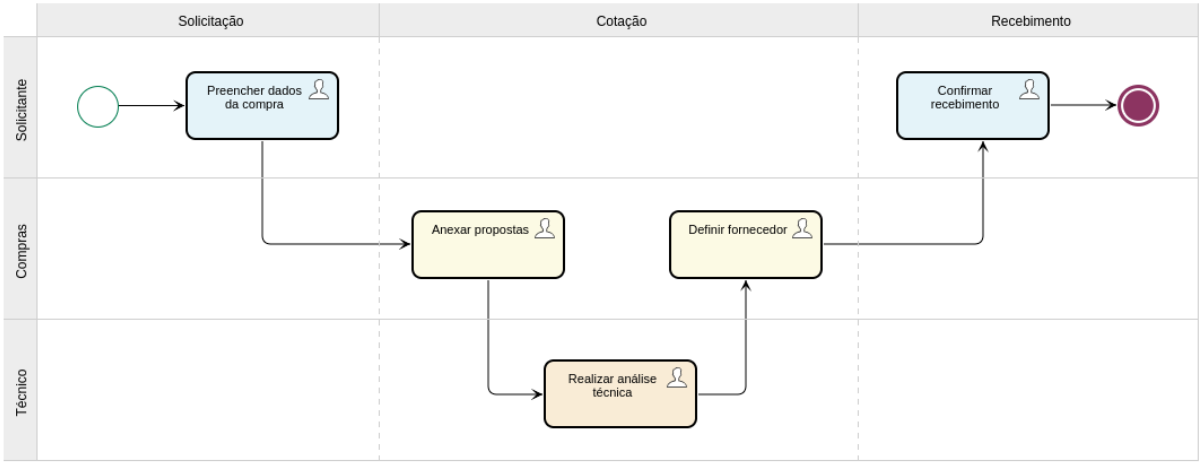
de todo o conteúdo necessário para a realização da tarefa, o instrutor se manteve disponível no Discord para sanar possíveis dúvidas.

Na etapa inicial da modelagem, foi preciso definir quem — e em quais condições — poderia utilizar o processo. É importante destacar que essa definição seria feita pelo cliente em uma situação real, mas em uma atividade didática é aceitável que essa definição seja feita pelo próprio desenvolvedor. Dessa forma, pude exercitar os conceitos de perfis de usuário utilizados nos processos e permissões de acesso dentro da plataforma através de grupos de usuários.

Em seguida, realizei a modelagem de dados, ou seja, criei os pacotes e classes necessárias para a elaboração do processo. A decisão de quais pacotes e classes criar foi embasada nos requisitos solicitados na atividade. Nessa etapa, além de aperfeiçoar minha habilidade de estruturar dados, também pude entender melhor as definições de objetos buscáveis e referências entre classes que o SYDLE ONE disponibiliza.

Por fim, atuei efetivamente na modelagem do processo de solicitação de compras. Esse foi o grande desafio da atividade, já que nesse momento tive o primeiro contato com o BPMN (Seção 2.10) e seus símbolos. Seguindo a documentação do SYDLE ONE, foram modeladas três versões do processo. Na primeira versão, foi elaborado um processo simples contendo apenas atividades de usuário, como pode ser visto na Figura 3.5.

Figura 3.5 – Versão 1.0 do processo Solicitação de Compra



Fonte: Do autor (2021).

Na versão seguinte, tive a oportunidade de utilizar os recursos do SYDLE ONE para automatizar um processo. Como pode ser visto no Apêndice B, essa automatização foi desenvolvida por meio de *gateways* que alteravam o fluxo de acordo com dados definidos pelo usuário.

Na última versão, uma melhoria foi realizada: o perfil de usuário apto a atender a atividade “Aprovar solicitação de compra (Alçadas)” tornou-se adaptável ao produto definido na solicitação de compra. Logo, o aprendizado também contemplou o mecanismo de papéis dinâmicos existente no SYDLE ONE. O diagrama da versão final pode ser visto no Apêndice C.

Ao finalizar a atividade, solicitei que o instrutor avaliasse a solução que desenvolvi. Foi verificado que o processo atendia todos requisitos solicitados e, após oito dias do início, o treinamento foi concluído.

3.3 O projeto

O projeto que integrei após o treinamento foi contratado por uma empresa privada para facilitar e modernizar sua atuação na venda e emissão de certificados digitais. Para evitar a exposição desse cliente, ele será mencionado neste trabalho utilizando a expressão “empresa contratante”.

Antes mesmo de me juntar à equipe, foi desenvolvida uma solução que percorre todas as etapas do serviço oferecido pela empresa contratante. Isto é, a solução abrange desde o *e-commerce* de certificados até a emissão e instalação deles nos dispositivos adequados.

A empresa contratante já realizava a emissão de certificados em sistemas legados. Contudo, a utilização de tecnologias antigas dificultava evoluções necessárias desses sistemas. Sendo assim, a migração para o SYDLE ONE foi responsável por agilizar adaptações às mudanças de regulamentação e, até mesmo, às novas oportunidades de mercado.

Através do módulo de BPM (Seção 3.1.1) do SYDLE ONE, os processos de emissão de certificado foram modelados com BPMN e automatizados com JavaScript. O mesmo aconteceu com outros processos, como o cadastro de operadores da empresa contratante.

Alguns recursos de CRM (Seção 3.1.1) do SYDLE ONE também foram aproveitados para gerenciar o relacionamento com clientes. As classes disponibilizadas para persistir dados de pessoas e empresas possibilitaram a criação de um processo de atendimento para oferecer suporte personalizado aos consumidores. Outras classes foram criadas para cadastrar e controlar as características de cada unidade de negócio da empresa contratante. Além disso, essas classes permitiam que as unidades administrassem algumas informações particulares, com seu horário de expediente e quadro de funcionários.

Outro componente importante da solução é a gestão de vendas, cobranças e catálogo de produtos. Por isso, o SYBOX de Billing (Seção 3.1.2) foi utilizado para realizar, além de outras demandas, o faturamento e a integração com diferentes meios de pagamento.

Também, foram produzidos alguns Analytics (Seção 3.1.1) para permitir a visualização de dados relevantes em tempo real e, consequentemente, contribuir com a tomada de decisões estratégicas. As unidades de negócio também se beneficiavam com o uso de Analytics, pois podiam consultar, do início ao fim do processo, o status atualizado das emissões.

Já o *front-end* do *e-commerce*, foi implementado utilizando HTML (Seção 2.3), Sass (Seção 2.7) e Angular (Seção 2.5), além de ser versionado com Git (Seção 2.8). Como o SYDLE ONE disponibiliza APIs configuráveis, ele foi utilizado como *back-end* da loja virtual. Ainda na aplicação *web*, foram desenvolvidos alguns serviços para permitir que o consumidor gerenciasse seus pedidos e instalasse seu certificado.

3.4 A equipe

Inicialmente, o projeto contava com um gerente, um analista de qualidade, uma *Product Owner* (PO) e, com minha entrada, 11 desenvolvedores. Diferentemente dos outros integrantes, o analista de qualidade e a PO eram colaboradores da empresa contratante. Assim, a priorização de demandas era feita pelo próprio cliente. Além disso, alguns conceitos do Scrum (Seção 2.1) eram utilizados no processo de *software*, como a divisão dos desenvolvimentos em *Sprints* e a realização de Planejamento de *Sprints*, *Dailies* e Retrospectiva.

Alguns meses depois, os gestores perceberam que a elevada quantidade de membros na equipe, além de atrapalhar a execução de ritos, não era a melhor forma de atender as necessidades do cliente. Mesmo que as demandas fossem priorizadas em diferentes categorias, como correção de *bugs* ou evoluções, elas concorriam pela atuação de desenvolvedores de um único time de propósito geral. Em vista disso, a equipe foi dividida em três *squads* com focos específicos: (1) Estruturante, (2) Evolutivo e (3) Sustentação. Ainda foram alocados mais alguns profissionais no projeto para garantir que as novas equipes fossem autossuficientes. A SYDLE disponibilizou um líder técnico, um arquiteto de *software* e uma analista de sucesso do cliente. Por fim, a empresa contratante contribuiu com mais dois analistas de qualidade e um PO.

O *squad* Estruturante era responsável por manter as plataformas do sistema atualizadas. Em outras palavras, esse time identificava e corrigia problemas estruturais do projeto, além de compatibilizar funcionalidades com novos recursos do SYDLE ONE e com atualizações das bibliotecas utilizadas no *front-end*. Esse grupo utilizava o Scrum como metologia de gerenciamento e era formado por um PO, um analista de qualidade e três desenvolvedores.

As evoluções de sistema demandadas pelo negócio ficava a cargo do *squad* Evolutivo. Portanto, esse time ocupava-se da criação de novas telas ou funcionalidades solicitadas pelo cliente. Esse *squad* também utilizava a metodologia Scrum e era composto por quatro desenvolvedores, um analista de qualidade, um líder técnico e um PO.

O *squad* Sustentação, em que fui alocado após a reestruturação, era encarregado por manter o sistema funcionando adequadamente. Apesar de realizar alguns ritos do Scrum, esse time se diferenciava dos demais por utilizar a metologia Kanban (Seção 2.2) como processo de *software*. Nessa frente, atuavam quatro desenvolvedores, um analista de qualidade, uma PO e a especialista em sucesso do cliente.

Apesar da divisão em três frentes, a equipe de projeto era única. Dessa forma, o arquiteto de *software* e o gerente do projeto atuavam em todos *squads*, como mostra a Figura 3.6.

Figura 3.6 – Equipe do projeto após reestruturação

Legenda: os blocos com laterais arredondadas indicam que os colaboradores são da empresa contratante, enquanto os retos representam profissionais da SYDLE.

Fonte: Do autor (2022).

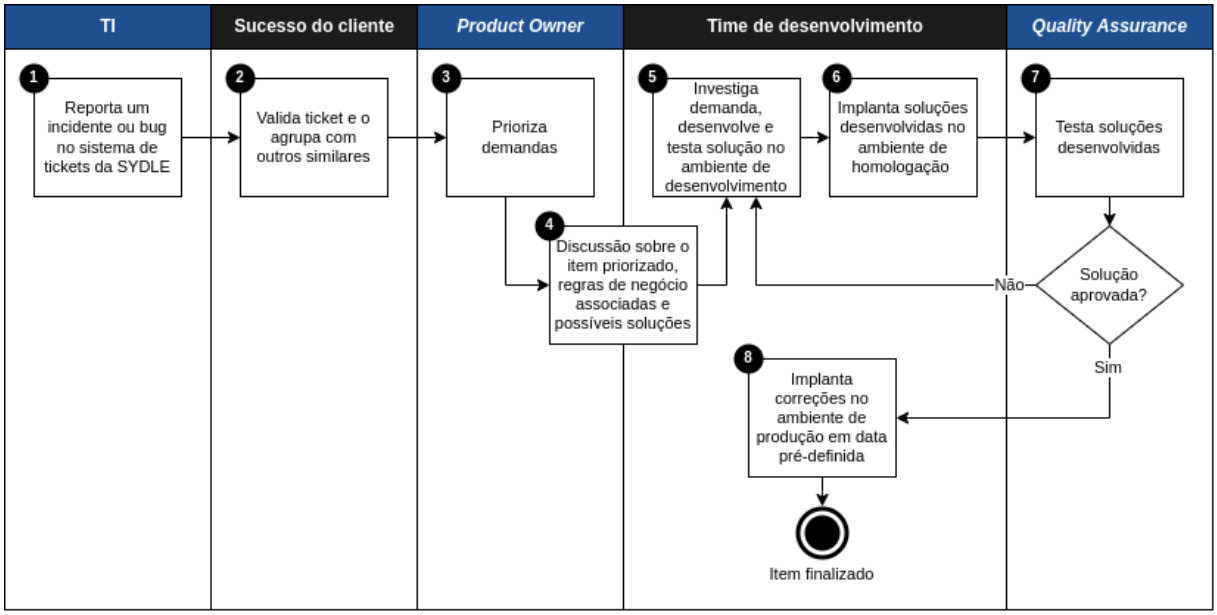
3.5 Processo de sustentação

Em geral, o *squad* Sustentação corrigia *bugs* e problemas inesperados encontrados em produção. Contudo, de forma mais rara, a equipe implementava algumas pequenas melhorias de sistema. Dada a necessidade de entregar soluções com frequência, utilizávamos a metodologia ágil Kanban para gerenciar o desenvolvimento de demandas. Além disso, para enriquecer o processo de *software*, também eram utilizados alguns elementos do Scrum: além de haver uma PO no time, realizávamos *Daily* e Retrospectiva. Também realizávamos uma reunião denominada *Grooming* — que não é prescrita pelo Scrum — para preparar as histórias de usuário. Todos os membros da equipe participavam da *Daily*, porém, como será abordado nessa seção, a participação na *Grooming* era exigida à PO, ao gerente do projeto e ao time de desenvolvimento. Ainda, utilizávamos *Sprints* de duas semanas para definir intervalos entre ritos e agendar entregas.

A Figura 3.7 ilustra de forma simplificada o ciclo de vida de uma demanda classificada como sustentação. Pode-se perceber que as atividades eram executadas por diferentes setores da empresa contratante e da SYDLE.

A primeira etapa da sustentação era feita pelo próprio cliente: a abertura de *tickets* no Service Desk (Seção 3.1.2). Majoritariamente, as demandas reportadas não degradavam o

Figura 3.7 – Fluxo de atividades desempenhadas no *squad* Sustentação



Legenda: O cabeçalho das raias indica o setor ou cargo responsável pelas atividades. Enquanto os cabeçalhos com fundo azul indicam que as atividades são desempenhadas pelas equipes da empresa contratante, os pretos indicam tarefas executadas pelo time SYDLE.

Fonte: Do autor (2022).

ambiente, isto é, não impediam seu uso. Por consequência, os *tickets* não eram resolvidos considerando uma fila de atendimento, mas passavam por estágios de priorização. Seguindo esse procedimento, a especialista em sucesso do cliente se encarregava de agrupar *tickets* similares e validar a riqueza de detalhes dos relatos. Eventualmente era solicitado ao cliente novas evidências para melhorar o entendimento do cenário. Por fim, após verificar a inexistência de implementações já em andamento para a demanda, o item era encaminhado à PO para registrá-lo como história no *backlog*.

Dando continuidade, a PO priorizava as demandas cadastradas no *backlog* e apresentava os itens selecionados aos desenvolvedores durante a reunião de *Grooming*. Esse evento, que ocorria pelo menos uma vez por *Sprint*, objetivava enriquecer o detalhamento das histórias através de discussões do time de desenvolvimento. Quando um item classificado como *bug* entrava em pauta, as condições de sua ocorrência eram demonstradas através das evidências extraídas do *ticket*. Assim, eu e os demais desenvolvedores realizávamos uma investigação para encontrar — ou nos aproximar — da origem do problema. Em todos os casos, com o auxílio da PO e do gerente de projeto, discutíamos as regras de negócio associadas à demanda e suas possíveis soluções.

No quadro Kanban, as histórias pautadas na *Grooming* eram categorizadas como aprovadas para implementação. A partir daí, os desenvolvedores assumiam a execução dessas histórias e, então, moviam-nas para a coluna de desenvolvimento. Vale destacar que não havia distinção entre as atribuições de um engenheiro de *software* efetivo e um estagiário. Isso significa que eu, mesmo sendo estagiário, me responsabilizava por realizar qualquer história aprovada, independentemente de seu nível de complexidade ou criticidade.

Em geral, o primeiro passo de uma implementação era aprofundar a investigação iniciada na *Grooming*. Por isso, era comum que eu solicitasse à PO novas evidências ou o agendamento de reuniões com os criadores do *ticket* originário da história. Nesses encontros, o solicitante da demanda compartilhava sua tela e executava as ações causadoras do problema para replicá-lo. Em adição, eu o instruía na realização de alguns procedimentos para gerar mais dados que auxiliassem minha análise. Então, munido das informações necessárias, eu elaborava uma proposta de solução e a encaminhava para aprovação da PO. Com a resposta positiva, ocorria, de fato, o desenvolvimento.

Ao finalizar a implementação, a história era movida para a coluna de testes do quadro Kanban. Nesse momento, um desenvolvedor diferente de mim — já que implementei a solução — verificava se a demanda foi resolvida. Caso algum teste falhasse, o testador me indicava os problemas remanescentes para que eu pudesse corrigi-los e, em seguida, repetir a etapa de testes. Por outro lado, caso os testes passassem, eu prosseguia com a implantação da solução no ambiente de homologação. Para isso, se houvessem modificações de modelagem dentro do SYDLE ONE, eu deveria gerar o SYBOX (Seção 3.1.2) contendo os itens modificados e aplicá-lo em homologação. De maneira similar, caso houvessem modificações no *front-end*, eu realizava o *merge* dos *branches* de desenvolvimento e de homologação para seguir com a implantação da aplicação *web* via Ansible[4]. Após a atualização do ambiente de homologação, eu solicitava que o especialista em qualidade da empresa contratante validasse a solução.

Assim como ocorria nos testes entre desenvolvedores, o testador da empresa contratante poderia aprovar a solução ou solicitar ajustes. Normalmente, o parecer dos testes era comunicado na *Daily*. Até obter essa aprovação final, repetiam-se as etapas de desenvolvimento, testes entre desenvolvedores, implantação em homologação e testes do especialista em qualidade.

Por fim, as histórias aprovadas nos testes estavam liberadas para subir para produção. Caso fosse uma correção pontual e, principalmente, em uma área de baixa criticidade, com-

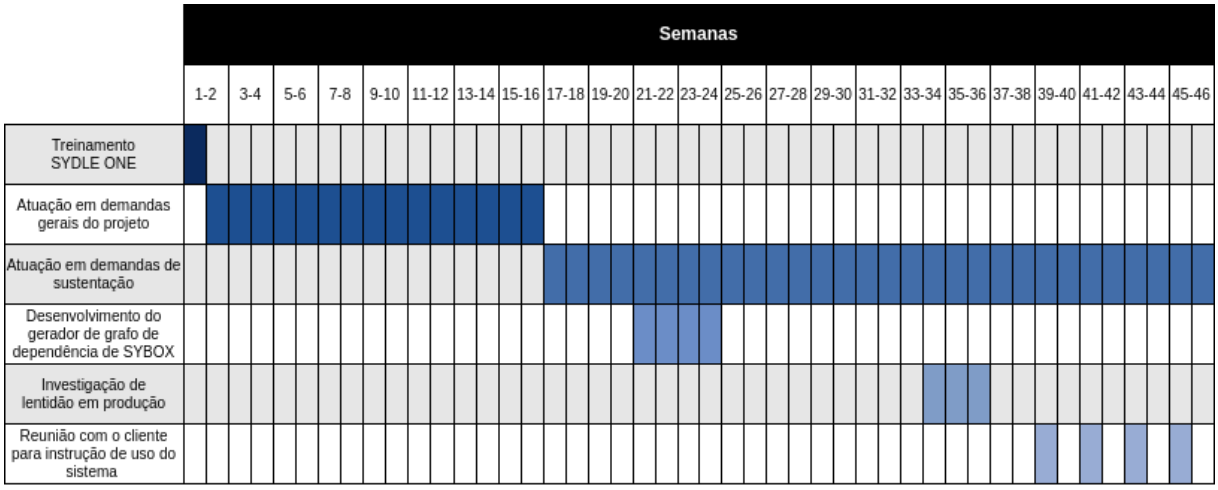
[4] Ansible é um *software* de gerenciamento de servidores e automatização de implantações. Isto é, essa ferramenta executa de forma automática o *build* da aplicação presente no *branch* pré-configurado.

binávamos com o cliente a implantação em um horário de uso moderado do sistema ou após a finalização do expediente. Em contrapartida, a implantação de itens mais delicados exigia um agendamento formal, além de ser realizada em conjunto com os outros *squads* e a equipe de qualidade da empresa contratante. Após a subida das soluções, os colaboradores da empresa contratante realizavam alguns testes manuais para validar o funcionamento satisfatório do sistema. Enfim, o ciclo de vida da demanda terminava.

3.6 Atividades desempenhadas

Durante minha participação no projeto, a maioria das atividades que desempenhei estava diretamente relacionada a correções ou implementações de regras de negócio específicas do cliente. Portanto, nesta seção, as atividades serão descritas de forma genérica para respeitar o sigilo de informações. O diagrama de Gantt visto na Figura 3.8 apresenta uma visão geral das minhas atuações no decorrer de 46 semanas participando do projeto.

Figura 3.8 – Diagrama de Gantt de atividades desempenhadas pelo autor



Fonte: Do autor (2022).

3.6.1 Treinamento do SYDLE ONE

A primeira semana do estágio foi dedicada à capacitação para o desenvolvimento no SYDLE ONE. Nesse treinamento, desenvolvi estruturas de dados na plataforma para, posteriormente, modelar e automatizar um processo corporativo de solicitação de compras. Enquanto a modelagem foi realizada utilizando BPMN, a automatização foi programada com JavaScript. Os detalhes do treinamento já foram descritos minuciosamente na Seção 3.2.

3.6.2 Atuação em demandas gerais do projeto

Quando fui alocado ao projeto de certificação digital da empresa contratante, a equipe era única e atendia todos os tipos de demanda do cliente. Por isso, inicialmente, trabalhei nas mais diversas áreas do projeto.

No *front-end*, utilizei — e aprimorei — meus conhecimentos de HTML, Angular e Sass para implementar a tela de instalação de certificados em dispositivos adequados. A partir de solicitações do cliente, também realizei algumas modificações de leiaute no *e-commerce*.

Já no SYDLE ONE, executei melhorias nos processos de emissão de certificados e de atendimento ao cliente. Nessas atividades, foi essencial o domínio de BPMN, JavaScript e Elasticsearch.

3.6.3 Atuação em demandas de sustentação

Após a reestruturação da equipe relatada na Seção 3.4, minhas atuações ficaram restritas às demandas de sustentação do sistema. Nesse contexto, as atividades que desempenhei se tornaram mais críticas, pois os problemas a serem resolvidos ocorriam em produção.

Durante esse período, criei um serviço JavaScript para melhorar a integração feita com o sistema de coleta biométrica utilizado no *front-end*. Também, atuei em evoluções do *e-commerce*, principalmente em correções e melhorias de validação de dados dos formulários.

No entanto, como as regras de negócio e automatizações estavam implementadas no SYDLE ONE, grande parte das demandas de sustentação estavam relacionadas à plataforma *low-code*. Em todas as demandas desse tipo que atuei, a primeira etapa realizada foi a investigação da causa do problema relatado pelos usuários. Posteriormente, utilizando BPMN, JavaScript e Elasticsearch, implementei as soluções que propus. As áreas de destaque dessas demandas foram os processos de venda, faturamento e emissão. Além disso, refatorei dois métodos importantes do sistema: o primeiro era utilizado para obter datas disponíveis para o atendimento ao cliente em unidades de negócio da empresa contratante e, o segundo, para atualizar CEPs[5] da base de dados do sistema.

Após adquirir a experiência necessária, fui escalado para realizar diversas implantações em produção. Antes do *deploy*, eu elaborava o roteiro de implantações e, com o auxílio de outros desenvolvedores, aplicava e testava os desenvolvimentos em um ambiente réplica de produção. Como a réplica era mantida no mesmo estado do ambiente de produção, os testes

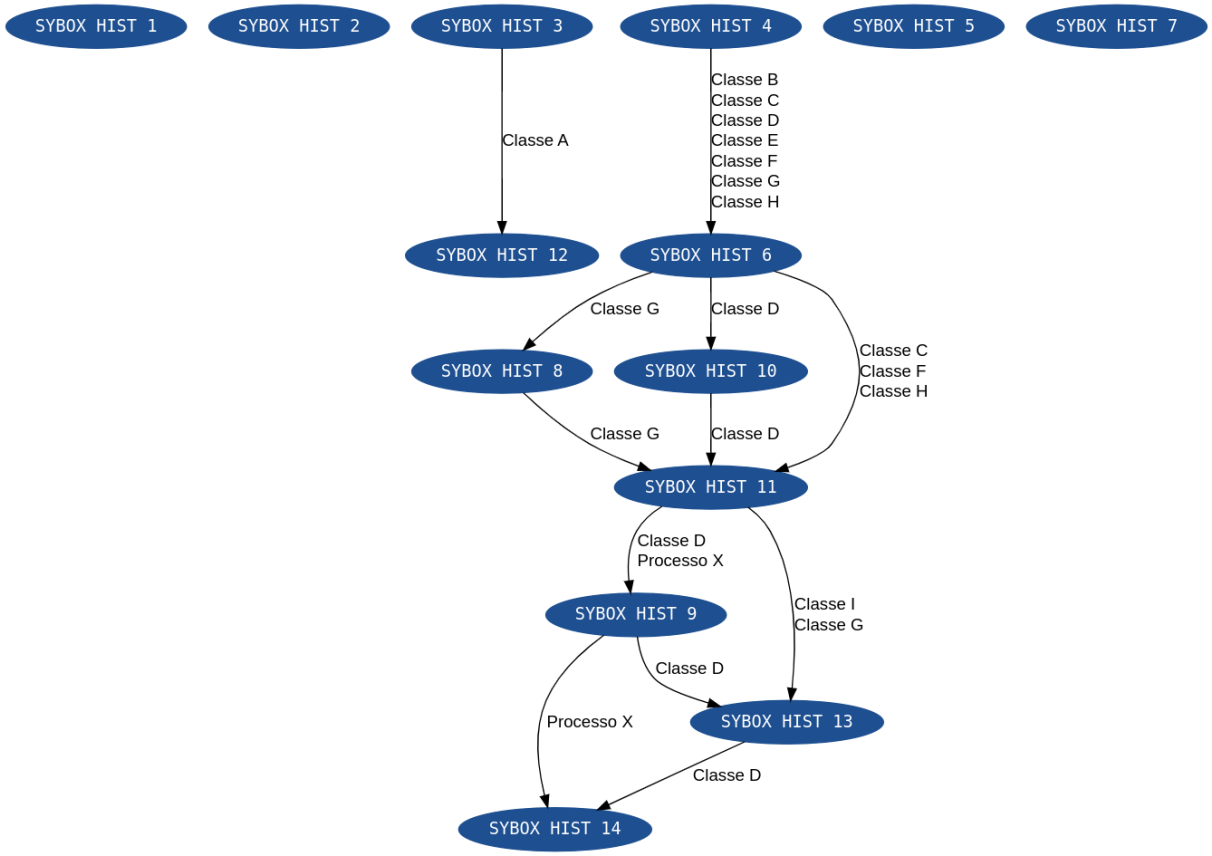
[5] Código de Endereçamento Postal

ajudavam a prevenir erros durante a implantação no ambiente final. Durante os *deploys*, eu era responsável pela aplicação de SYBOX e correções de erros que, por ventura, tenham passado despercebido nas etapas anteriores.

3.6.4 Desenvolvimento do gerador de grafo de dependência de SYBOX

Após minha primeira participação como observador em um *deploy*, sugeri aos gestores do projeto a criação de um gerador de grafo de dependências de SYBOX. Esse grafo nos indicaria a ordem correta de aplicação de SYBOX e, consequentemente, evitaria a sobrescrita de modificações. Assim, após a aprovação de meus superiores, me dediquei ao desenvolvimento desse recurso. A Figura 3.9 exibe um exemplo genérico de um grafo gerado pela ferramenta.

Figura 3.9 – Exemplo de grafo de dependência de SYBOX



Legenda: Cada nó do grafo representa o SYBOX de uma história aprovada para implantação em produção. Quando dois SYBOX estão conectados, significa que existe dependência entre eles. O rótulo da aresta indica quais artefatos estão presentes simultaneamente nos dois SYBOX conectados. Na origem da aresta está o SYBOX criado há mais tempo, portanto ele deve ser aplicado antes do SYBOX da ponta da aresta.

Fonte: Do autor (2022).

Para produzir o grafo, elaborei um algoritmo que monta uma matriz de adjacências dos objetos de SYBOX selecionados. Posteriormente, gerei a representação visual utilizando a biblioteca Viz.js[6].

Visto que a implementação do gerador foi feita de maneira genérica, a ferramenta pode ser utilizada em qualquer projeto da SYDLE. Dessa forma, como sugerido pelo meu gestor, apresentei a solução a desenvolvedores de outras equipes para que eles pudessem utilizá-la.

3.6.5 Investigação de lentidão em produção

Em um determinado momento, os usuários relataram que algumas funcionalidades estavam demorando mais do que o habitual para serem executadas. Prontamente, os gestores do projeto organizaram uma equipe para investigar a causa do baixo desempenho do sistema. Além de profissionais da área de infraestrutura da SYDLE, eu e outro desenvolvedor fomos relacionados para atuar nessa investigação.

Para buscar a origem da lentidão, nos reuníamos virtualmente em horários estratégicos e utilizávamos os sistemas de monitoramento do SYDLE ONE para detectar os gargalos de execução. No decorrer de três semanas, todos os problemas foram identificados e solucionados. Vale ressaltar que eu e o outro desenvolvedor fomos responsáveis por propor as soluções dos problemas relacionados ao desenvolvimento de *software*.

3.6.6 Reunião com o cliente para instrução de uso do sistema

Nos últimos meses de projeto, eu e outro desenvolvedor fomos escalados para participar de reuniões semanais com o cliente para sanar dúvidas de utilização da plataforma. Como não era necessária a presença de dois desenvolvedores na reunião, a cada semana eu e outro desenvolvedor alternávamos a participação.

Nesses encontros, o cliente apresentava suas dificuldades ao usar o sistema e, caso fosse possível, eu o instruía a executar as ações corretamente. Via de regra, eu tomava notas dos casos apresentados e, havendo necessidade, eu sugeria aos gestores do projeto a realização de melhorias ou correções.

[6] Viz.js é uma biblioteca de código aberto que renderiza grafos definidos em linguagem específica.

4 CONCLUSÃO

Em sua maioria, as disciplinas da graduação oferecem atividades práticas que, pelo caráter didático, nem sempre se aproximam da complexidade encontrada em projetos do mundo real. Assim, minha participação no estágio supervisionado na SYDLE foi a oportunidade de aplicar, em desafios reais, os conhecimentos que adquiri na universidade.

Durante o estágio, os fundamentos de diversas disciplinas foram essenciais para executar minhas atribuições. Primeiramente, o conteúdo aprendido sobre lógica e técnicas de programação na disciplina de Introdução aos Algoritmos foi o alicerce para atuar no desenvolvimento de *software*. Da mesma maneira, dada as características arquiteturais do SYDLE ONE, as habilidades adquiridas na disciplina de Estruturas de Dados me permitiram avaliar e desenvolver as melhores estruturas para a criação de classes de dados e para elaboração de algoritmos. As manutenções que realizei no *front-end* exigiam as noções de HTML e JavaScript aprendidas na disciplina de Programação *Web*. No dia a dia de trabalho, as disciplinas de Engenharia de *Software* e Processos de *Software* facilitaram meu entendimento do gerenciamento do projeto e das metodologias ágeis utilizadas. Já a disciplina de Algoritmos em Grafos me permitiu abstrair dados em estruturas de grafos que facilitavam análises necessárias.

Ainda, no estágio pude observar na prática a flexibilidade das metodologias ágeis. Diferentemente da forma como é praticado na universidade, alguns conceitos de Scrum e outros de Kanban foram misturados para atender às necessidades do projeto que participei.

Não menos importante, a vivência com outros profissionais possibilitou um intercâmbio de conhecimentos indispensáveis para meu crescimento profissional. Até mesmo o contato com especialistas de outras áreas, como infraestrutura, foi aproveitado para ampliar minha compreensão das tecnologias que fogem ao escopo do desenvolvimento de *software*.

Em resumo, no estágio, não só minhas habilidades técnicas foram aperfeiçoadas, mas também as interpessoais. Da mesma forma que a atuação em uma empresa privada não é capaz de construir uma base sólida de conhecimentos teóricos, na graduação não é possível vivenciar os desafios reais do mercado de trabalho. Dessa forma, fica claro que os aprendizados adquiridos na universidade e a experiência prática oferecida pelo estágio se complementam para a formação de profissionais de excelência.

Ao fim da minha participação no projeto de certificação digital, meu contrato de estágio foi renovado. Dessa forma, fui alocado em outro projeto contratado por um banco estatal

brasileiro. Por fim, a empresa manifestou interesse na minha contratação em definitivo após a conclusão do curso.

4.1 Sugestões de melhoria

Para terminar, serão sugeridas propostas de melhoria concebidas a partir da minha experiência. Primeiramente, é importante implementar algum algoritmo de ordenação topológica para o gerador de grafo de dependência de SYBOX (Seção 3.6.4). Atualmente, só é possível visualizar a ordem de aplicação dos SYBOX pela representação gráfica das dependências, cabendo ao usuário ordenar os itens manualmente. A automatização dessa operação, por um algoritmo conhecido, evitaria a ocorrência de erros humanos.

Em relação ao desenvolvimento no SYDLE ONE, a plataforma não conta com um recurso nativo para documentação de métodos. Porém, programadores que atuam na plataforma criaram uma solução que percorre os métodos de uma classe para interpretar anotações JSDoc[1] e estruturá-las em um documento HTML. Apesar de útil, o modelo gerado é baseado em documentação para APIs *web*, contendo informações que fogem ao escopo do SYDLE ONE. Portanto, é importante que a plataforma desenvolva uma funcionalidade nativa para produzir documentações coesas que facilitem e mantenham organizados os desenvolvimentos no SYDLE ONE.

Por fim, foi perceptível durante o estágio a importância do uso de *frameworks* na programação *web*. Contudo, não há disciplinas no curso de graduação que capacite bem os alunos para o uso dessas tecnologias tão exigidas pelas empresas. Por isso, sugere-se que sejam ofertados cursos entre semestres que abordem tecnologias em alta no mercado, como o *framework* Angular citado neste trabalho, para que alunos interessados possam se preparar melhor para concorrerem às vagas de trabalho.

[1] JSDoc é uma API para geração de documentações de sistemas JavaScript utilizando comentários inseridos no próprio código-fonte (JSDOC, 2019).

REFERÊNCIAS

ANDERSON, D. J. **Kanban**: Successful evolutionary change for your technology business. Blue Hole Press, 2010. ISBN 9780984521401. Disponível em: <https://books.google.com.br/books?id=RJ0VUkfUWZkC>.

ANGULAR. **CLI Overview and Command Reference**. 2016. Disponível em: <https://angular.io/cli>. Acesso em: 27 jul. 2022.

ANGULAR. **Built-in directives**. 2022. Disponível em: <https://angular.io/guide/built-in-directives>. Acesso em: 27 jul. 2022.

ANGULAR. **Dependency injection in Angular**. 2022. Disponível em: <https://angular.io/guide/dependency-injection>. Acesso em: 27 jul. 2022.

ANGULAR. **What is Angular?** 2022. Disponível em: <https://angular.io/guide/what-is-angular>. Acesso em: 26 jul. 2022.

BOCK, A. C.; FRANK, U. Low-code platform. **Business & Information Systems Engineering**, Springer Science and Business Media LLC, v. 63, n. 6, p. 733–740, nov. 2021. Disponível em: <https://doi.org/10.1007/s12599-021-00726-8>.

CHACON, S.; STRAUB, B. **Pro Git**. 2. ed. Berlim, Alemanha: APress, 2014. Disponível em: <https://github.com/progit/progit2/releases/download/2.1.350/progit.pdf>.

ELASTIC. **Data in**: documents and indices. 2019. Disponível em: <https://www.elastic.co/guide/en/elasticsearch/reference/current/documents-indices.html>. Acesso em: 01 ago. 2022.

ELASTIC. **Information out**: search and analyze. 2019. Disponível em: <https://www.elastic.co/guide/en/elasticsearch/reference/current/search-analyze.html>. Acesso em: 01 ago. 2022.

ELASTIC. **O que é o Elasticsearch?** 2019. Disponível em: <https://www.elastic.co/pt/what-is/elasticsearch>. Acesso em: 01 ago. 2022.

ELASTIC. **What is Elasticsearch?**: You know, for search (and analysis). 2019. Disponível em: <https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html>. Acesso em: 01 ago. 2022.

GIT. **Distributed**. 2012. Disponível em: <https://git-scm.com/about/distributed>. Acesso em: 11 ago. 2022.

GIT. **Git**. 2012. Disponível em: <https://git-scm.com>. Acesso em: 11 ago. 2022.

ICET-UFLA. **Curso de Graduação em Ciência da Computação (Bacharelado)**. 2021. Disponível em: <https://icet.ufla.br/graduacao/ciencia-computacao-bacharelado>. Acesso em: 12 ago. 2022.

JSDOC. **Getting Started with JSDoc 3**. 2019. Disponível em: <https://jsdoc.app/about-getting-started.html>. Acesso em: 11 ago. 2022.

MDN WEB DOCS. **Como funciona o CSS**. 2020. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/CSS/First_steps/How_CSS_works>. Acesso em: 31 jul. 2022.

MDN WEB DOCS. **Como CSS é estruturado**. 2021. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/CSS/First_steps/How_CSS_is_structured>. Acesso em: 31 jul. 2022.

MDN WEB DOCS. **JavaScript - Glossário**. 2021. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Glossary/JavaScript>. Acesso em: 20 jul. 2022.

MDN WEB DOCS. **Sobre JavaScript**. 2021. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/About_JavaScript>. Acesso em: 20 jul. 2022.

MDN WEB DOCS. **Uma reintrodução ao JavaScript (Tutorial de JS)**. 2021. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/A_re-introduction_to_JavaScript>. Acesso em: 20 jul. 2022.

OMG. **Business Process Model and Notation (BPMN)**: Version 2.0.2. 2013. Disponível em: <https://www.omg.org/spec/BPMN/2.0.2/PDF>.

OMG. **Graphical notations for business processes**. 2018. Disponível em: <https://www.omg.org/bpmn/index.htm>. Acesso em: 01 ago. 2022.

SASS. **Documentation**. 2013. Disponível em: <https://sass-lang.com/documentation>. Acesso em: 31 jul. 2022.

SASS. **Sass Basics**. 2013. Disponível em: <https://sass-lang.com/guide>. Acesso em: 31 jul. 2022.

SASS. **Dart Sass**. 2018. Disponível em: <https://sass-lang.com/dart-sass>. Acesso em: 31 jul. 2022.

SASS. **sass:math**. 2019. Disponível em: <https://sass-lang.com/documentation/modules/math>. Acesso em: 31 jul. 2022.

SASS. **Values**. 2019. Disponível em: <https://sass-lang.com/documentation/values>. Acesso em: 31 jul. 2022.

SCHWABER, K. Scrum development process. In: SUTHERLAND, J. et al. (Ed.). **Business Object Design and Implementation**. Londres: Springer London, 1997. p. 117–134. ISBN 978-1-4471-0947-1.

SCHWABER, K.; SUTHERLAND, J. **The 2020 Scrum Guide**. [S.l.], 2020. Disponível em: <https://scrumguides.org/scrum-guide.html>. Acesso em: 31 jan. 2022.

SEVERANCE, C. Javascript: Designing a language in 10 days. **Computer**, v. 45, n. 2, 2012.

SYDLE. **Analytics**: Informação em tempo real. 2022. Disponível em: <https://www.sydle.com/br/analytics>. Acesso em: 08 ago. 2022.

SYDLE. **Billing**: Gestão de cobranças recorrentes. 2022. Disponível em: <https://www.sydle.com/br/billing/?plataforma-gestao-cobrancas-recorrentes>. Acesso em: 08 ago. 2022.

SYDLE. **BPM**: Automação de processos. 2022. Disponível em: <https://www.sydle.com/br/bpm/?plataforma-automacao-processos>. Acesso em: 08 ago. 2022.

SYDLE. **Clientes**. 2022. Disponível em: <https://www.sydle.com/br/clientes/>. Acesso em: 12 ago. 2022.

SYDLE. **CRM**: Relacionamento com clientes. 2022. Disponível em: <https://www.sydle.com/br/crm/?plataforma-relacionamento-clientes>. Acesso em: 08 ago. 2022.

SYDLE. **Service Desk**: Portal de serviços. 2022. Disponível em: <https://www.sydle.com/br/service-desk/?plataforma-portal-servicos>. Acesso em: 08 ago. 2022.

SYDLE. **SYBOX**: Soluções corporativas. 2022. Disponível em: <https://www.sydle.com/br/sybox>. Acesso em: 08 ago. 2022.

SYDLE. **SYDLE ONE | Plataforma de BPM, ECM, CRM, Service Desk e Billing.** 2022. Disponível em: <https://www.sydle.com/br/>. Acesso em: 08 ago. 2022.

VALENTE, M. T. **Engenharia de Software Moderna**: Princípios e práticas para desenvolvimento de software com produtividade. Belo Horizonte: [s.n.], 2020. Disponível em: <https://engsoftmoderna.info>. Acesso em: 31 jan. 2022.

W3C. **HTML & CSS - W3C**. 2016. Disponível em: <https://www.w3.org/standards/webdesign/htmlcss.html>. Acesso em: 25 jul. 2022.

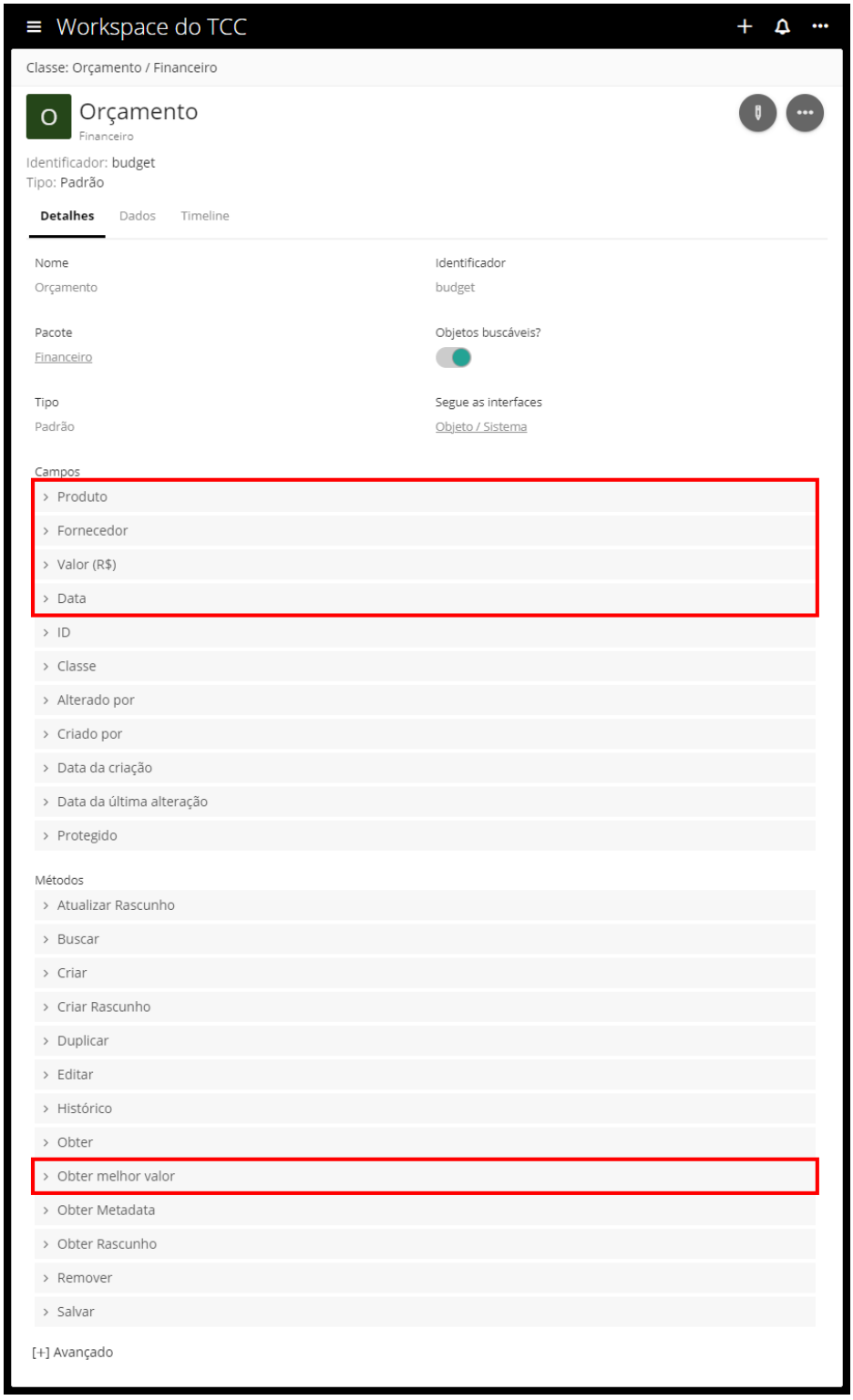
W3C. **W3C and WHATWG to work together to advance the open Web platform**. 2019. Disponível em: <https://www.w3.org/blog/2019/05/w3c-and-whatwg-to-work-together-to-advance-the-open-web-platform>. Acesso em: 25 jul. 2022.

W3SCHOOLS. **CSS Introduction**. 2007. Disponível em: <https://www.w3schools.com/css/css_intro.asp>. Acesso em: 31 jul. 2022.

W3SCHOOLS. **CSS Syntax**. 2017. Disponível em: <https://www.w3schools.com/css/css_syntax.asp>. Acesso em: 31 jul. 2022.

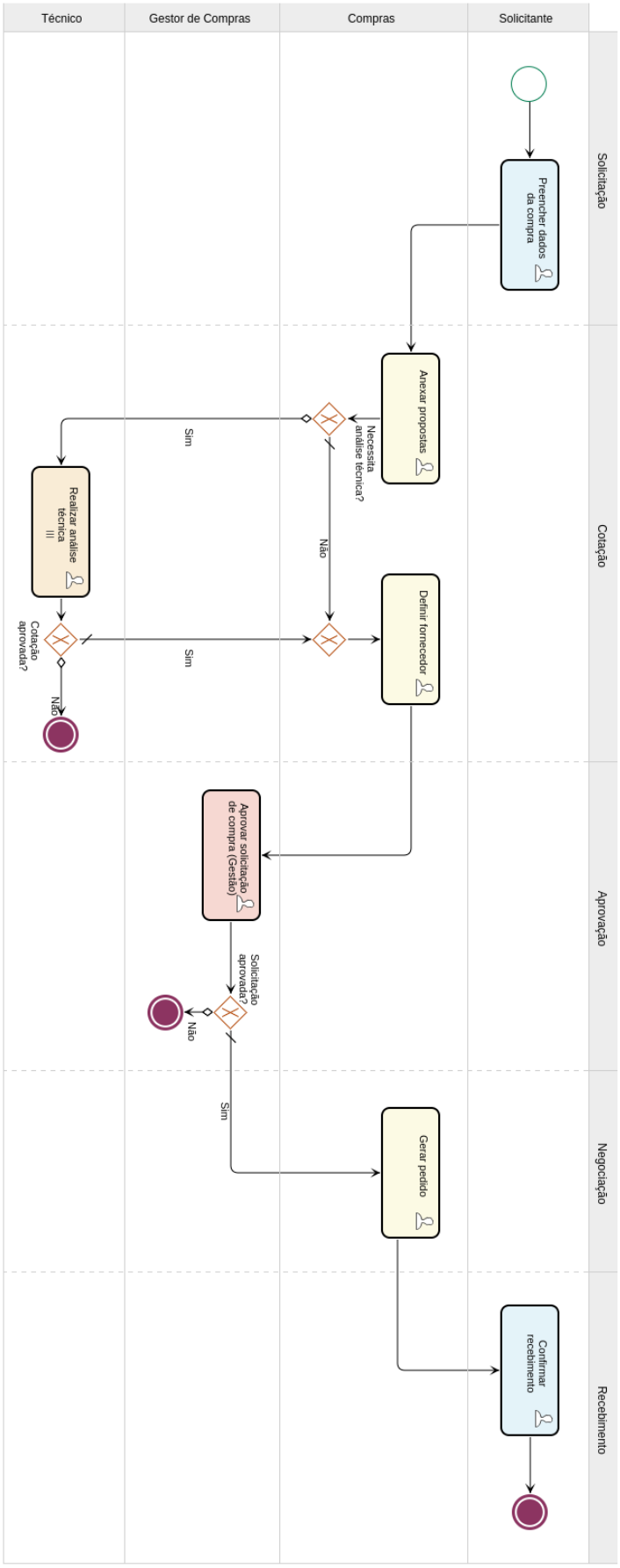
WHATWG. **HTML Living Standard**. 2022. Disponível em: <https://html.spec.whatwg.org>. Acesso em: 25 jul. 2022.

APÊNDICE A – Classe “Orçamento” modelada no SYDLE ONE



Os campos e métodos contornados de vermelho foram criados pelo autor. Os não contornados são gerados automaticamente pelo SYDLE ONE no momento da criação da classe.

APÊNDICE B – Versão 2.0 do processo Solicitação de Compra



APÊNDICE C – Versão 3.0 do processo Solicitação de Compra