GABRIEL HENRIQUE SILVA BERNARDO

A UTILIZAÇÃO DO DESIGN SYSTEM PARA O DESENVOLVIMENTO DE PRODUTOS NA IOASYS

LAVRAS – MG

2022

GABRIEL HENRIQUE SILVA BERNARDO

A UTILIZAÇÃO DO DESIGN SYSTEM PARA O DESENVOLVIMENTO DE PRODUTOS NA IOASYS

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Bacharelado em Ciência da Computação para a obtenção do título de Bacharel.

Prof. Dr. Maurício Ronny de Almeida Souza
Orientador

LAVRAS – MG

2022

GABRIEL HENRIQUE SILVA BERNARDO

A UTILIZAÇÃO DO DESIGN SYSTEM PARA O DESENVOLVIMENTO DE PRODUTOS NA IOASYS

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Lavras, como parte das exigências do Curso de Bacharelado em Ciência da Computação para a obtenção do título de Bacharel.

Prof. Dr. Maurício Ronny de Almeida Souza	UFLA
Prof. Dr. Paulo Afonso Parreira Junior	UFLA
Nathália Stéphany Nascimento	IOASYS

Prof. Dr. Maurício Ronny de Almeida Souza
Orientador

LAVRAS – MG
2022

Dedico esse trabalho aos meus pais Maria Helena e Ricardo Edson que sempre me incentivaram.

AGRADECIMENTOS

Agradeço, primeiramente, a Deus pois sempre me guiou e me ajudou durante toda essa jornada da minha vida. Agradeço aos meus pais que sempre me incentivaram e não me deixaram desistir, mesmo nos momentos mais difíceis que tive nessa jornada acadêmica. Durante toda minha vida, eles estiveram comigo e me proporcionaram tudo o que eu precisei para ter um educação de qualidade. Tenho muito a agradecer aos meus amigos que ganhei durante esse tempo de graduação e também aos que já faziam parte da minha vida antes disso e que sempre me incentivaram a continuar e não desistir, me ajudaram a estudar e até mesmo a conversar e ouvir em momentos difíceis que surgiram nessa caminhada. Agradeço também a minha namorada por estar sempre ao meu lado me ajudando e me incentivando em tudo durante essa jornada. E, por fim, agradeço aos meus professores que me trouxeram tanto conhecimento durante todo esse tempo de graduação que, sem eles, eu não estaria aqui hoje, em especial ao meu orientador, Maurício Ronny de Almeida Souza, que teve paciência e me incentivou nessa reta final com meu estágio e a escrita deste trabalho.

“Cada sonho que você deixa pra trás, é um pedaço do seu futuro que deixa de existir.” (Steve Jobs)

RESUMO

Uma empresa de desenvolvimento de software lida todos os dias com diversas pessoas e seja clientes ou colaboradores e gerenciar tudo isso não é algo fácil. Esse relatório de estágio supervisionado tem por fim relatar o processo de desenvolvimento de um design system que está sendo utilizado para projetos internos da empresa Ioasys, tanto para agilizar o desenvolvimento de novos sistemas como também trazer um padrão para seus produtos internos. O estagiário também participou do desenvolvimento do primeiro sistema a utilizar esse design system que foi o Vision, um sistema criado para ajudar os colaboradores da área de experiência a gerir seus colaboradores e seus clientes de uma forma mais simples e funcional. Para isso, as principais tecnologias utilizadas foram React Js, TypeScript e Storybook.

Palavras-chave: Design System. Desenvolvimento web. React Js. TypeScript

ABSTRACT

The abstract should contain representative words of the work content, located below the abstract, separated by two spaces, preceded by the keyword expression. These representative words are spelled with the first letter capitalized, separated by point.

Keywords: Design System. Web developer. React Js. TypeScript

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A apresentação de sistemas é um fator decisivo na experiência do usuário. Um sistema com uma boa usabilidade e um design intuitivo são fortes pontos para a escolha entre um sistema X e Y, assim, a usabilidade não é só fazer "sistemas bonitos", é também pensar sempre em quem vai utilizá-lo e implementar algo que o usuário goste de usar no seu dia a dia e que também seja fácil. Entretanto, garantir a padronização visual de sistemas pode ser dificil devido a diversas ferramentas disponíveis para se implementar um sistemas e também pelas diversas formas que é possível fazê-lo. Mesmo que se utilize uma única tecnologia os desenvolvedores pensam diferente e implementam sistemas que diferem-se um do outro, seja por facilidade com em determinadas metodologias ou pela sua criatividade.

Neste sentido, os *design systems* visam principalmente a padronização de sistemas, seja para uma identidade visual de uma empresa, ou seja para simplesmente facilitar um desenvolvedor. O uso de um *design system* pode favorecer, e muito, uma empresa de desenvolvimento de software, pensando primeiramente nas vantagens que um padrão de design pode trazer, como a padronização de todos os produtos da empresa o que traz familiaridade para quem o consome e com isso vem a facilidade de aprender a usar um novo software que utiliza o mesmo *design system*.

Além disso, um *design system* traz para os desenvolvedores uma grande economia de tempo já que os componentes estão prontos para serem utilizados e permitem aos desenvolvedores focarem mais em funcionalidades ao invés de gastar tempo refazendo componentes antes já feitos. Assim, este documento descreve as atividades desenvolvidas pelo autor no estágio supervisionado realizado na empresa Ioasys, no período de 5 meses. O principal objetivo do estágio foi o desenvolvimento de um *design system*, o Legacy, para apoiar o desenvolvimento de soluções internas da organização e sua consequente aplicação no desenvolvimento do Vision, um sistema para para gestão e organização de colaboradores e projetos da empresa.

1.1 Sobre a ioasys

A Ioasys [1] é uma empresa de inovação tecnológica, especializada em trazer o melhor do mundo digital para seus clientes, ajudando as marcas a inovar através de transformação digital e princípios de metodologias ágil. Com seu lema "De pessoas para Pessoas", a Ioasys tem sua cultura fortemente enraizada que nos é passado para novos membros desde o primeiro dia na empresa e que se mantém vivo durante todo o tempo que eles atuarem lá. A empresa preza muito o bem estar de seus colaboradores e com isso transformam o local de trabalho em um lugar prazeroso e acolhedor, e por isso hoje ela é detentora do prêmio de *Great Place to Work*, fornecida para empresas que assim como a Ioasys se preocupam não apenas com seus clientes mais também com seus colaboradores.

Atualmente a empresa conta com cerca de 400 colaboradores presentes em mais de 100 cidades do Brasil e do mundo, Seu maior escritório fica localizado em Belo Horizonte em Minas Gerais, porém conta também com escritórios em Lavras, Minas Gerais e São Paulo capital. Neste contexto, o estagiário atuou remotamente na equipe responsável pelo Vision, onde atuava junto com mais três desenvolvedores, uma scrum master e 1 designer.

1.2 Objetivos

Neste contexto, o objetivo deste documento é descrever as atividades desenvolvidas pelo autor durante o estágio supervisionado realizado na empresa Ioasys. O objetivo do estágio tinha como primeiro passo, desenvolver um sistema interno voltado para ajudar as pessoas com o gerenciamento de projetos e de colaboradores. Com esse sistema o pessoal da gerência teria uma visualização mais completa do que as planilhas oferecem hoje em dia, e para o desenvolvimento desse sistema surgiu a oportunidade de desenvolver um Design System para projetos internos da Ioasys e com isso manter um padrão entre todos e acelera o desenvolvimento do sistema de Gerência como de novos projetos futuros. O estágio foi realizado no período de Setembro de 2021 à Abril/2022, totalizando 582 horas.

[1] https://ioasys.com.br/

1.3 Organização do Trabalho

O restante deste trabalho segue a seguinte organização. O Capítulo 2 apresenta uma descrição das principais tecnologias utilizadas durante o estágio. No Capítulo 3 são apresentadas as atividades desenvolvidas no estágio, com ênfase na concepção, desenvolvimento e aplicação do Design System "Legacy"; No Capítulo 4 são apresentadas as considerações finais deste trabalho.

2 CONCEITOS E TECNOLOGIAS

Neste capítulo serão apresentadas as tecnologias utilizadas para o desenvolvimento do estagiário durante seu período de estágio na empresa Ioasys, como principais tecnologias utilizadas se destacam as de desenvolvimento *web*, focadas principalmente no desenvolvimento *front-end* que foi a área de atuação do mesmo.

2.1 Desenvolvimento WEB

A internet trouxe para o dia a dia das pessoas novos hábitos de entretenimento, comportamento, comunicação e consumo. O Desenvolvimento de aplicações *Web* precisa acompanhar essas mudanças na rotina dos usuários, com a utilização de métodos e ferramentas específicas (MILETTO; BERTAGNOLLI, 2014). Desenvolvimento *Web* é a área da tecnologia voltada para construção de sites, *softwares*, aplicativos e quaisquer outras ferramentas voltadas para a internet. A área do desenvolvimento *Web* é composta por duas subáreas, que são: *front-end* e *back-end*, sendo o *back-end* a área voltada para infraestruturada de um site e por outro lado o *front-end* é responsável pela experiência do usuário, ou seja, as telas por onde os usuários irão interagir com o sistema (MILETTO; BERTAGNOLLI, 2014).

Nas subseções a seguir, são descritas as principais tecnologias utilizadas para o desenvolvimento de *front-end* utilizadas pelo estagiário.

2.1.1 JavaScript

O JavaScript (JS) é uma linguagem interpretada e baseada em objetos como funções de primeira classe (MOZILLA, 2022d). O JS é amplamente conhecido como linguagem de *script* para páginas *web*, porém bastante utilizada em vários outros ambientes sem navegadores também, como Node JS, Apache CouchDB e Adobe Acrobat (MOZILLA, 2022d). JavaScript é uma linguagem baseada em protótipos, multi-paradigma e dinâmica, suportando estilos de orientação a objeto, imperativo e declarativos (MOZILLA, 2022d). No desenvolvimento de aplicações *web*, o JavaScript é muito utilizado junto com HTML (*HyperText Markup Language*) e CSS (*Cascading Style Sheets*) no *front-end*. O JavaScript é responsável por mudanças que ocorrem na página (eventos) enquanto o HTML e o

CSS são responsáveis pela estruturação e apresentação dos conteúdos. A utilização do JS em páginas HTML se dá pela inserção da *tag* **<script>**. A Figura 2.1 mostra um exemplo simples da utilização do JavaScript em uma página HTML. A *tag* **<script>** dispara um alerta que mostra o texto *"Hello World"* para o usuário. Durante o período de estágio, foi utilizado o JavaScript apenas para treinamento. Em desenvolvimentos posteriores foi utilizado por padrão o TypeScript (ver seção 2.1.4) por fornecer algumas vantagens que serão citadas posteriormente.

Figura 2.1 – Exemplo de "Hello World"em JavaScript.

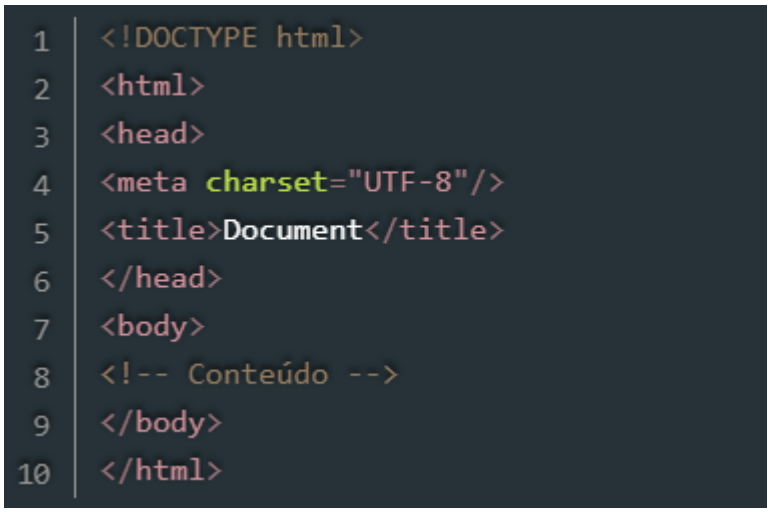
Fonte: https://ricardo-reis.medium.com/hello-world-em-javascript-61fc5f8fc59c

2.1.2 HTML - Linguagem de Marcação de Hipertexto

O HTML (*HyperText Markup Language* ou Linguagem de Marcação de HiperTexto) é o bloco de construção mais básico da *web* (MOZILLA, 2022c). O HTML geralmente é utilizado com o CSS, para descrever a aparência, e com o JavaScript, para descrever suas funcionalidade e comportamentos. O HTML usa a “marcação” para anotar textos, imagens, tabelas e outros conteúdos para exibição em um navegador da *Web* (MOZILLA, 2022c). A Figura 2.2 apresenta a estrutura básica de uma página

desenvolvida em HTML. Os elementos HTML são separados por "*tags*", que consistem no nome do elemento entre "< "e "> ". No exemplo, a *tag* <html> engloba a aplicação como um todo, e é essa tag que informa ao navegador que todo conteúdo posterior deve ser interpretado como códigos HTML. Um documento HTML é dividido em dois blocos principais, demarcados pelas *tags* <head> e <body>. A tag <head> representa o cabeçalho da página, onde são adicionadas informações importantes, como por exemplo o título da página (usando a tag <title>), ou também metadados (usando a tag <meta>). No exemplo, os metadados informam qual a coleção de caracteres o navegador deve utilizar para analisar a página. A *tag* <body> é responsável pelo conteúdo da página como texto, imagens, tabelas ou qualquer outro conteúdo a ser implementado no site.

Figura 2.2 – Estrutura básica de uma página HTML



Fonte: https://www.devmedia.com.br/html-basico-codigos-html/16596

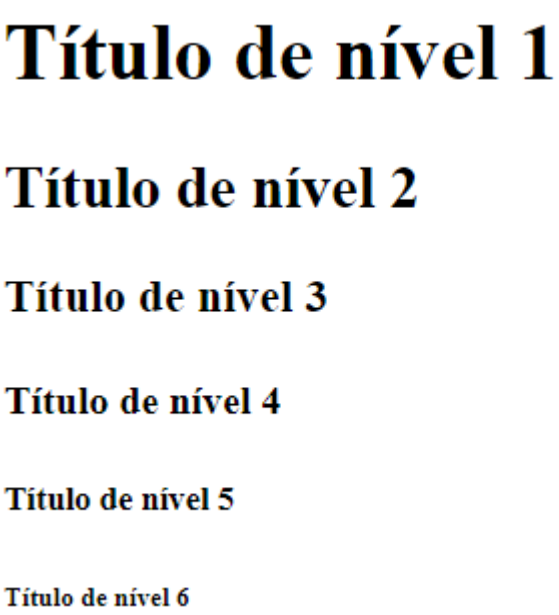
O HTML utiliza hierarquia na sua estrutura ainda mais quando se diz respeito a cabeçalhos, utilizados para descrever brevemente os tópicos da seção, as *tags* que tem essa função são as <h1> até o <h6>, sendo o <h1> o nível de seção mais alto e <h6> o mais baixo, como demonstrado na Figura 2.3 e Figura 2.3 (MOZILLA, 2022b).

Figura 2.3 – Diferentes níveis de cabeçalho.

```
<h1>Título de nível 1</h1>
<h2>Título de nível 2</h2>
<h3>Título de nível 3</h3>
<h4>Título de nível 4</h4>
<h5>Título de nível 5</h5>
<h6>Título de nível 6</h6>
```

Fonte: Autor

Figura 2.4 – Resultado dos diferentes níveis de cabeçalho.



Fonte: Autor

Durante o tempo em que o estagiário atuou na empresa, HTML "puro"não era frequentemente utilizado, entretanto para o desenvolvimento *web* a utilização do mesmo é essencial, e a biblioteca *React Js* (ver Seção 2.1.5), utiliza-se uma variação das *tags* do HTML.

2.1.3 CSS - Folhas de Estilo em Cascata

O CSS (*Cascading Style Sheets* ou Folhas de Estilo em Cascata) é uma linguagem de estilo onde podemos descrever como os elementos de um código HTML serão apresentados na tela e customizar seus comportamentos padrões como cores, fontes e *layouts* (MOZILLA, 2022a). Sem essa estilização, o conteúdo de páginas HTML são renderizados de uma forma padrão e geralmente listados um abaixo do outro na ordem que foram escritos, conforme o exemplo da Figura 2.5 e Figura 2.6. O CSS define regras de estilo para os elementos HTML, podendo ser usado diretamente como parâmetros dentro das *tags* HTML, ou importadas através de arquivos de estilos (arquivos com final .css). Na Figura 2.7 e Figura 2.8 são apresentados, respectivamente, um trecho de código CSS e o resultado da estilização quando aplicado à pagina apresentada na Figura 2.6.

Figura 2.5 – Exemplo de código HTML de uma página web.

HTML

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8"/>
<title>Document</title>
</head>
<body>
<h1>Título do meu Site</h1>
<h2>Subtítulo do meu Site</h2>
<img src="https://images.pexels.com/photos/5474295/pexels-photo-5474295.jpeg?auto=compress&cs=tinysrgb&dpr=2&h=750&w=1260" alt="Figura de um computador" title="Figura do meu site"/>
</body>
</html>
```

Fonte: Autor

Figura 2.6 – Resultado do código da figura 2.5.

Fonte: Autor

O CSS é capaz de alterar completamente o visual de um site, como mostrado na Figura 2.7 foi aplicada 3 regras de estilização para a página anterior, a primeira regra de estilo foi a responsável por aplicar a cor azul na *tag* de título que neste exemplo foi representada pela *tag* <*h1*>, para definir uma cor é possível referenciá-la pelo nome no caso de cores mais comuns como por exemplo cores primárias (Azul, verde, vermelho, amarelo e etc), porém caso seja necessário a utilização de uma cor mais específica também é possível referenciá-la pelo seu código HEX ou RGB. A segunda regra é a responsável pela *tag* de subtítulo que neste caso foi utilizado a *tag* <*h2*>, aplicando a cor vermelha como estilização, e para ambas as *tags* (<*h1*> e <*h2*>) também foi modificada a fonte utilizada para Roboto. Para a última regra foi feito o redimensionamento da imagem para que tivesse uma melhor visibilidade. E com todas as regras aplicadas é possível ver o resultado na Figura 2.8.

Durante o desenvolvimento de atividades no período de estágio, o estagiário utilizou bastante o CSS para estilizar não só os sistemas feitos como também os componentes do *design system*.

Figura 2.7 – CSS utilizado para estilização.

```css
h1 {
color: blue;
font-family: 'Roboto', sans-serif;
}

h2 {
  color: red;
  font-family: 'Roboto', sans-serif;
}

img {
  height: 30%;
  width: 30%;
}
```

Fonte: Autor

Figura 2.8 – Resultado do HTML da figura 2.8.

Título do meu Site

Subtítulo do meu Site

Fonte: Autor

2.1.4 TypeScript

O TypeScript é uma extensão do JavaScript voltada para facilitar o desenvolvimento de aplicativos JavaScript em larga escala. O TypeScript oferece um sistema de classes, módulos, interfaces e um sistema bastante completo de tipagem (BIERMAN; ABADI; TORGERSEN, 2014). A maior diferença do TypeScript quando comparado com o JavaScript é possível ver nas Figuras 2.9 e 2.10 onde a diferença mais chamativa é a tipagem, que deve ser passada seguida de ":"logo após o nome da variável criada. Na Figura 2.10 podemos ver que as tipagens são respeitadas pelo interpretador da própria IDE (neste exemplo a IDE utilizada foi Visual Studio Code [1]) que nos alertar a não atribuir um número em uma variável antes declarada como *String*, diferente do JavaScript por não ser uma linguagem fortemente tipada não retorna nem um tipo de alerta ou erro.

Figura 2.9 – Declarações de variáveis em JavaScript.

```
//JavaScript
let age = 23; // Variavel do tipo Number

let firtName = "Gabriel"; // Variavel do tipo String

let fruits = ["apple", "orange", "pineapple"]; // Variavel do tipo Array de String

age = "23";
firtName = 23;
fruits = 23;
```

Fonte: Autor

Figura 2.10 – Declarações de variáveis em TypeScript.

```
//TypeScript
let age: Number = 23; // Variavel do tipo Number

let firtName: String = "Gabriel"; // Variavel do tipo String

let fruits: Array<String> = ["apple", "orange", "pineapple"]; // Variavel do tipo Array de String

age = "23";    O tipo 'string' não pode ser atribuído ao tipo 'Number'.
firtName = 23;    O tipo 'number' não pode ser atribuído ao tipo 'String'.
fruits = 23;    O tipo 'number' não pode ser atribuído ao tipo 'String[]'.
```

Fonte: Autor

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

Por padrão a empresa que o estagiário atuou utiliza-se no lugar do JavaScript a linguagem de programação TypeScript.

2.1.5 React Js

A principal tecnologia de desenvolvimento na empresa em que o estagiário atuou por padrão foi o *React JS* porém como citado acima foi definido a utilização do React junto do TypeScript em vez do JavaScript. O React é uma biblioteca de JavaScript que seu foco maior é em componentes encapsulados o que gera códigos mais limpos, sem repetição de blocos de códigos, com uma manutenção bastante fácil e com uma rapidez de desenvolvimento muito grande, por permitir cria um componente uma única vez e utilizá-lo em vários locais do seu sistema (META PLATFORMS, 2022). O React utiliza uma forma de criar elementos bastante interessante chamada JSX, tem uma sintaxe bem simples e parecida com o HTML o'que traz uma familiaridade para novos desenvolvedores, porém o JSX não é interpretado pelo navegador o'que faz necessário a utilização de um "transcompilador" para fazer essa conversão que neste caso utiliza-se o Babel, porém não se faz necessário a escrita dos componentes necessariamente utilizando o JSX também é possível escrevê-los em JavaScript puro, podemos ver a diferença das duas formas de escrever um componente pelas Figuras 2.11 e 2.12.

Durante o período que o estagiário atuou na empresa, a utilização do React esteve presente a todo momento, foi utilizado para desenvolver tantos os componentes do *design system* (Legacy), como o sistema de gestão de funcionários (*vision*) (META PLATFORMS, 2022).

Figura 2.11 – Simples componente feito em React utilizando JSX.

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Olá, {this.props.name}!
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Taylor" />,
  document.getElementById('hello-example')
);
```

Fonte: https://pt-br.reactjs.org/

Figura 2.12 – Mesmo componente da Figura 2.11 porém feito em JS puro

```
class HelloMessage extends React.Component {
  render() {
    return React.createElement(
      "div",
      null,
      "Ol\xE1, ",
      this.props.name,
      "!"
    );
  }
}

ReactDOM.render(React.createElement(HelloMessage, { name: "Taylor" }),
document.getElementById('hello-example'));
```

Fonte: https://pt-br.reactjs.org/

2.2 Design System

Um *design system* é a representação programática de um *website* criado geralmente por um designer. É uma coleção de cores, fontes, botões, imagens, estilos, ritmos tipográficos e padrões UI usados para transmitir significado e intenção (GODBOLT, 2016).

A compreensão do objetivo de um *design system* é o primeiro passo para implementar uma solução adequada que ajude as equipes com futuros produtos. Uma implementação clara e bem definida de um *design system* deixa os desenvolvedores e designers com mais tempo para concentrar em resolver as necessidade dos clientes, em vez de recriar elementos e reinventar soluções (VESSELOV; DAVIS, 2019). Existem seis áreas que interligadas constituem um *design system*, são elas:

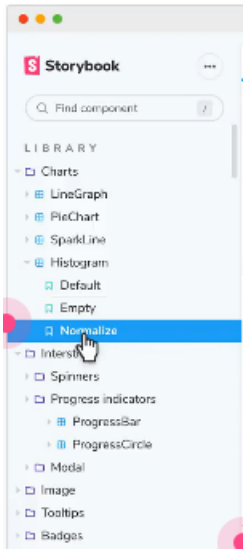
- Layout: Medidas definidas que constituem espaçamentos e sistemas de *grids*;
- Componentes: Elementos centrais de uma interface. Estes incluem botões, e campos de formulário;
- Regiões: Paradigmas de design abrangentes, tais como navegação ou pesquisa;
- Estilos: Aspectos centrais de linguagem visual. Estes incluem cores, tipografia e iconografia;
- Conteúdo: Informações sobre a voz e o tom, como as diretrizes de pontuação. O conteúdo também pode incluir terminologia, se o produto a ser desenvolvido tiver vocabulário específico;
- Usabilidade: Regras que definem a acessibilidade e internacionalização (VESSELOV; DAVIS, 2019).

O *design system* surge quando começamos pensar em páginas *web* modulares, com as tecnologias que temos hoje para desenvolver sistemas *webs* como React JS ou Vue podemos parar de pensar em páginas inteiras e começar a pensar em sistemas modulares, essas tecnologias nos permitem criar componentes que podemos usar em várias páginas do projeto e com isso o desenvolvimento fica mais rápido, e esse é o principal conceito de um *design system* porém visando reutilização não apenas no mesmo sistema. Durante o tempo em que o estagiário atuou na empresa o desenvolvimento de um *design system* veio cobrir a falta de padronização da empresa de produtos internos, como também o tempo que economizaria e a facilidade que traria para os desenvolvedores que fossem atuar em projetos internos futuramente.

2.2.1 Storybook

O StoryBook é uma ferramenta muito utilizada pelos desenvolvedores *front-end* e designers da empresa que o estagiário atuou, todos os dias na criação de novos componentes e principalmente para o desenvolvimento do *design system* o StoryBook sempre estava presente. O StoryBook é uma ferramenta muito útil principalmente para o React que utiliza a ideia de componentização, ele é responsável por simplificar a documentação e os testes dos componentes implementados. Com poucas linhas de código temos uma documentação rica e detalhes totalmente reativa, ou seja se um componente tem variações documentadas elas podem ser testadas na tela do StoryBook e ter um *feedback* visual do componente sendo renderizado e modificado sem ao menos ter que chamá-lo externamente (THE MIT LICENSE, 2022). O StoryBook cria para seu sistema uma página isolada para cada componente implementado ou facilita sua visualização e seus testes de implementação, e todos os componentes são listados na barra lateral do mesmo como podemos ver na Figura de exemplo 2.13.

Figura 2.13 – Menu de seleção de componetes o Storybook.



Fonte: https://storybook.js.org/

2.3 Desenvolvimento Ágil

Em fevereiro de 2001, dezessete profissionais da área da programação se reuniram para pensar em algo para melhorar a forma de desenvolver *softwares* e assim surgiu o movimento ágil, uma declaração de valores e princípios para desenvolver um *software* (JIM HIGHSMITH, 2001). E após isso vem se adotando cada vez mais nas empresas de *software* utilização de metodologias ágeis , também conhecidos como métodos leves são formas de desenvolver produtos, comumente utilizados em cenários que devem ser apresentados com uma certa frequência uma entrega de resultados e ou quando ocorrem com uma certa frequência alterações nos requisitos do produto.(CARVALHO; MELLO, 2012).

2.3.1 Scrum

Scrum é uma estrutura leve que ajuda pessoas, equipes e organizações a criar valor por meio de soluções adaptáveis para problemas complexos. *Scrum* é simples, e o *framework Scrum* é intencionalmente incompleto, definindo apenas as partes necessárias para implementar a teoria *Scrum*. O *Scrum* é construído sobre a inteligência coletiva das pessoas que o utilizam. As regras do *Scrum* não fornecem instruções detalhadas às pessoas, mas orientam seus relacionamentos e interações.(SCHWABER; SUTHERLAND, 2020).

Na empresa que o estagiário atuou utiliza-se como estrutura de organização das equipes o *Scrum* que é um método de desenvolvimento ágil e que ajuda a trabalhar em equipe e sempre aprender e melhorar com as experiências de ciclo de desenvolvimento que no *Scrum* leva o nome de *sprint*. Um *Sprint* é um curto período de tempo, onde é definido quais tarefas irão ser desenvolvidas, e com isso vem um controle melhor do que pretende ser desenvolvido e também uma divisão do "todo"em micro tarefas o que facilita tanto na divisão de trabalhos como nas entregas.

2.3.2 Planejamento e execução

O método *Scrum* foi planejado para trabalho em equipe, seja seguindo ele, como foi feito, ou também com algumas alterações para adequar melhor a cada equipe, como por exemplo tamanho da *sprint*. Para projetos mais extensos não compensa *sprints* pequenas pois as entregas não seriam

finalizadas no tempo determinado, como também para projetos menores, os *sprints* grandes deixaram os desenvolvedores ociosos por um tempo até o final da *sprint*. No projeto em que o estagiário atuou, por ter algumas tarefas menores, eram feitas *sprints* de 12 dias úteis, o que era suficiente para atuar nas atividades propostas. Antes de começar a *sprint*, há alguns "ritos"a serem seguidos que serão descritos abaixo.

2.3.3 Sprint Planning

A *Planning*, como o nome sugere, é o rito de planejamento da nossa *sprint*, ou seja, um rito com uma duração um pouco extensa pois é nela que vão ser divididas as tarefas do "todo"para micro tarefas e então planejar o que os desenvolvedores podem fazer no tempo definido da *sprint*, quais tarefas tem impedimento de outras, quanto tempo estimamos para atuar em cada tarefa, e definir também a data dos próximos ritos e do final da da *sprint*.

2.3.4 Daily Scrum

As *dailys* são os encontros diários e rápidos, pois o ideal é não passar dos 15 minutos para não atrasar o desenvolvimento de ninguém. As reuniões geralmente eram feitas pela manhã na empresa que o estagiário atuava, para ver o andamento das tarefas e ver em qual cada um está atuando, e caso haja algum impedimento discutimos para tentar resolver.

2.3.5 Sprint Review

Ao final do período determinado para o fim da *sprint*,existem dois ritos a serem seguidos: o primeiro trata-se da *Review* em que vão ser apresentar tudo o que foi desenvolvido durante esse tempo, mostrando os códigos, interfaces e uma pequena apresentação de slides se achar necessário. É nela também que se pode conversar sobre algum impedimento que possa ter acontecido ou imprevisto que não deixou finalizar alguma tarefa proposta, e essa apresentação geralmente é feita para o cliente também porém o projeto que o estagiário atuou era interno então a apresentação era feita apenas para a *Scrum Master*.

2.3.6 Sprint Retro

E por final temos a Retro que é a última reunião do ciclo de uma *sprint*, na equipe que o estagiário atuou era feito por costume a *Review* junto com a Retro no mesmo dia por ser um projeto interno e pequeno, mais geralmente são ritos distintos que dependendo do projeto pode ser bastante longos. Na retro é discutido como a equipe trabalhou naquela *sprint*, se teve algum problema com alguma coisa que envolva a equipe, ou se aconteceu algo externo que atrasou o desenvolvimento, como também era levantado o que deu certo para ser aplicado nas próximas *sprints*.

Figura 2.14 – Ciclo de execução de uma Sprint.

Fonte: https://www.atlassian.com/br/agile/scrum/sprints

3 DESENVOLVIMENTO DO LEGACY: UM DESIGN SYSTEM PARA CRIAR A IDENTIDADE AOS PROJETOS DA IOASYS

Neste capítulo são descritas as atividades realizadas pelo estagiário no período de 5 meses, na organização Ioasys. Durante este período o estagiário foi responsável pelo desenvolvimento de um *design system*, o Legacy, para apoiar o desenvolvimento padronizado dos componentes *front-end* das aplicações Web da empresa. O Legacy foi utilizado para o desenvolvimento do sistema Vision, um sistema de gestão de funcionários e projetos da empresa.

As seções a seguir descrevem os princípios e motivações para a concepção do Legacy (Seção 3.1), o detalhamento do seu desenvolvimento (Seção 3.2), e sua aplicação no desenvolvimento do sistema Vision (Seção 3.3).

3.1 Concepção do Legacy

O Legacy é um *design system* projetado para satisfazer a necessidade de criação de uma identidade visual única e o estabelecimento de um conjunto de componentes reutilizáveis para todos os produtos desenvolvidos internamente pela Ioasys. O foco é apoiar o desenvolvimento de aplicações Web e Mobile com a biblioteca React. Os princípios que nortearam a concepção do Legacy são a simplicidade e a facilidade de uso. Por “simplicidade” espera-se que o Legacy seja um *design system* que não traga dificuldade na hora da sua utilização, que venha com uma familiaridade a todos programadores que já utilizam pacotes NPM (Node Package Manager [1]). Por "facilidade de uso", espera-se que sejam implementadas regras de organização e categorização de componentes, de forma que os desenvolvedores possam localizar e aplicar componentes de forma rápida e intuitivamente.

O projeto do Legacy foi inspirado em um *design system* de código aberto chamado Jota DS [2]. O objetivo do Jota DS é fornecer uma base reutilizável de diversos componentes para desenvolvimento de *design system*, assim sendo, quando um designer for começar a desenvolver o seu próprio *design system* ter uma base de componentes essenciais e padrões a serem seguidos para se ter um *design system* que satisfaça quem for utilizá-lo. Um padrão de projeto utilizado do Jota DS que foi incorporado

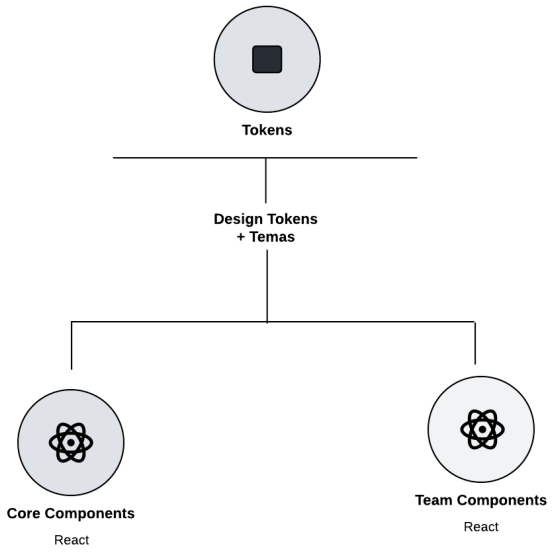
[1] https://https://www.npmjs.com/
[2] https://jota.meiuca.co/

ao Legacy é os "*Design Tokens*". Design Tokens são definições de padrões dos menores parâmetros como cores, espessuras de linhas, tamanhos de fontes, espaçamentos e etc, e com isso podemos forçar que os componentes tenham coerência entre si. (MEIUCA, 2022)

Dessa forma, a arquitetura dos *Desing Tokens* do Legacy foi estruturada em *tokens*, *layouts*, tipografia e ícones. Assim, *tokens* são *Design Tokens* referente às regras de estilo para trazer coerência e harmonia entre os componentes. Nessa esteira, os *layouts* são *Design Tokens* referentes às regras para os grandes componentes ou até mesmo as páginas que utilizam componentes do Legacy. Assim, a tipografia corresponde à *Design Tokens* referentes de padrões que os textos dos componentes devem seguir. Logo, os ícones são *Design Tokens* referentes à padronização dos ícones a serem utilizados em todo o projeto.

A arquitetura do Legacy é apresentada na Figura 3.1.

Figura 3.1 – Diagrama da Arquitetura do Legacy



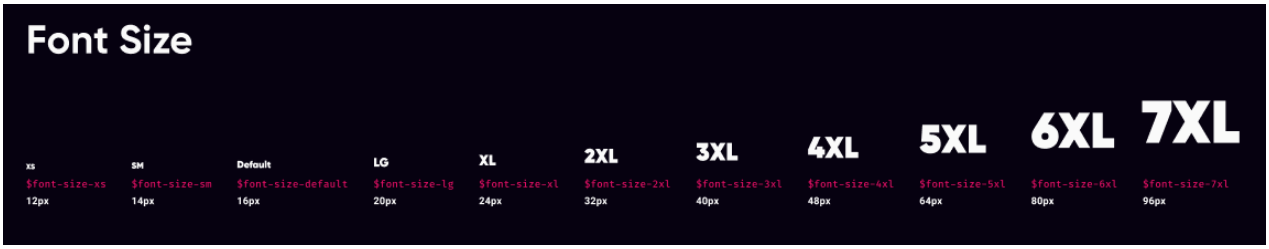
Fonte: Autor

As subseções a seguir descrevem detalhadamente cada um dos *Desing Tokens* que compõem o Legacy, separada em *tokens*, *layouts*, tipografia e ícones.

3.1.1 Tokens

Os Tokens trazem para o projeto do Legacy algumas regras de estilo que deixam os componentes coerentes entre si e funcionais, pensando na experiência do usuário. A título de exemplo, é possível citar o tamanho das fontes que foram utilizadas no Legacy, que são nomeadas e documentadas como mostrado na Figura 3.2, para que o desenvolvedor, no momento de estilizar o componente, possa apenas passar o nome do texto que deseja e, com isso, manter um padrão em todos os componentes. Com isso facilitando a alteração caso necessário, pois como os tamanhos são variáveis a alteração é feita apenas em um lugar e será aplicado para todo o sistema.

Figura 3.2 – Token de Font Size.



Fonte: Ioasys

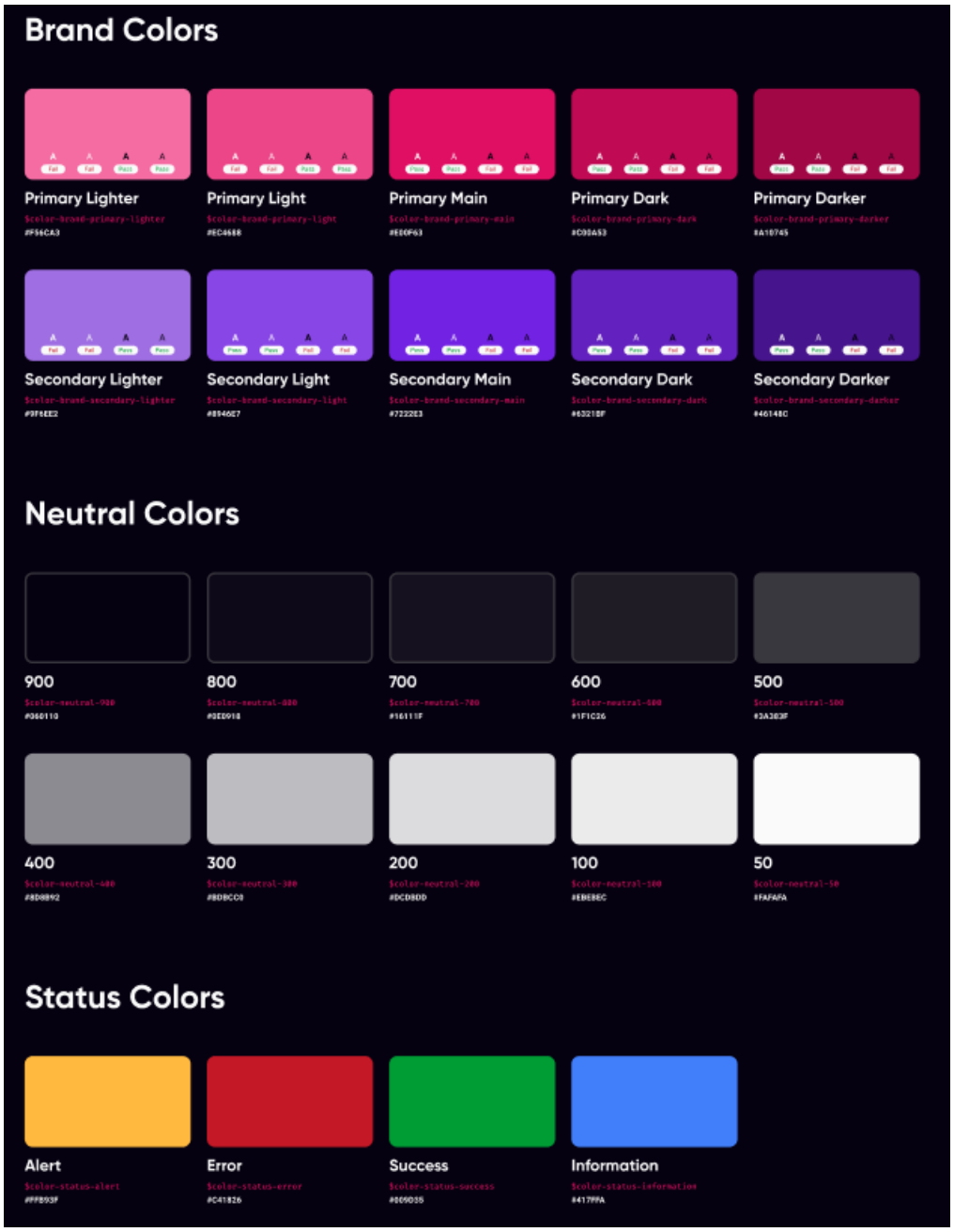
Ainda na parte de tokens, há os documentos além do *Font Size* (Tamanho da fonte) citado acima, como o *Line Height*(Altura da linha), *Fixed Line Height* (Altura de linha fixa), *Border Radius* (Raio da borda), *Border Width* (Largura da borda), *Opacity* (Opacidade), *Shadows* (Sombras), *Background Blur* (Desfoque de fundo) e *Spacing* (Espaço). Assim, todos esses padrões são representações de medidas pré-definidas, como mostrado no *Font-Size*. Além disso, foi definido também algumas características das fontes a serem utilizadas no projeto, como por exemplo a família da fonte e a espessura da fonte, como mostrado na Figura 3.3. Elas são divididas em duas, primária e secundária, em que cada uma tem sua família específica, porém, ambas compartilham a espessura da fonte.

Figura 3.3 – Token de Font Family e Font Weight.

Fonte: Ioasys

Como exposto na Figura 3.4, as cores definidas para os componentes do Legacy estão separadas em *Brand Colors* (Cores da marca), que serão cores mais marcantes e que trazem uma identidade visual para o design system; e as *Neutral Colors* (Cores neutras) que, como o nome já diz, são cores mais neutras que foram utilizadas para fundos e/ou fontes; e, por fim, os *Status Colors* (Cores de status), que foram utilizadas em alguns componentes para demonstrar o status de disponibilidade ou para passar alguma informação mais visual de erro ou sucesso, por exemplo.

Figura 3.4 – Token de Colors.

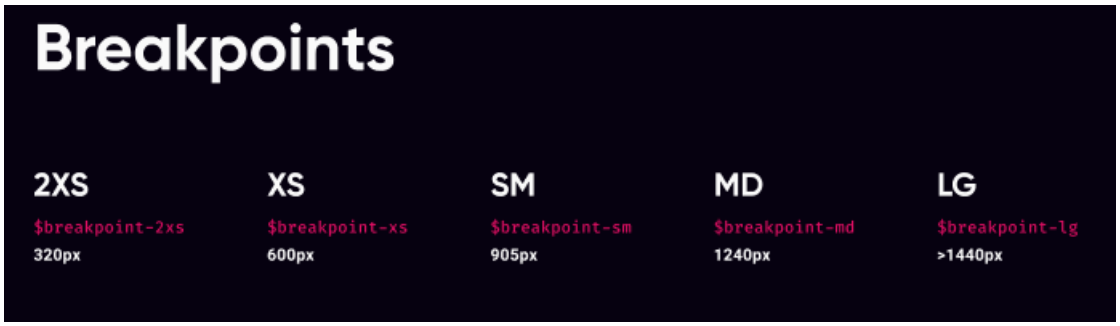


Fonte: Ioasys

3.1.2 Layouts

Em *Layout*, foram definidos alguns padrões voltados para os grandes componentes ou, até mesmo, páginas que utilizam os componentes do Legacy. Isto é, nessa área foram definidos os *Breakpoints* (Pontos de interrupção), que são as medidas para mudar o *layout* de uma aplicação responsiva como mostrado na Figura 3.5.

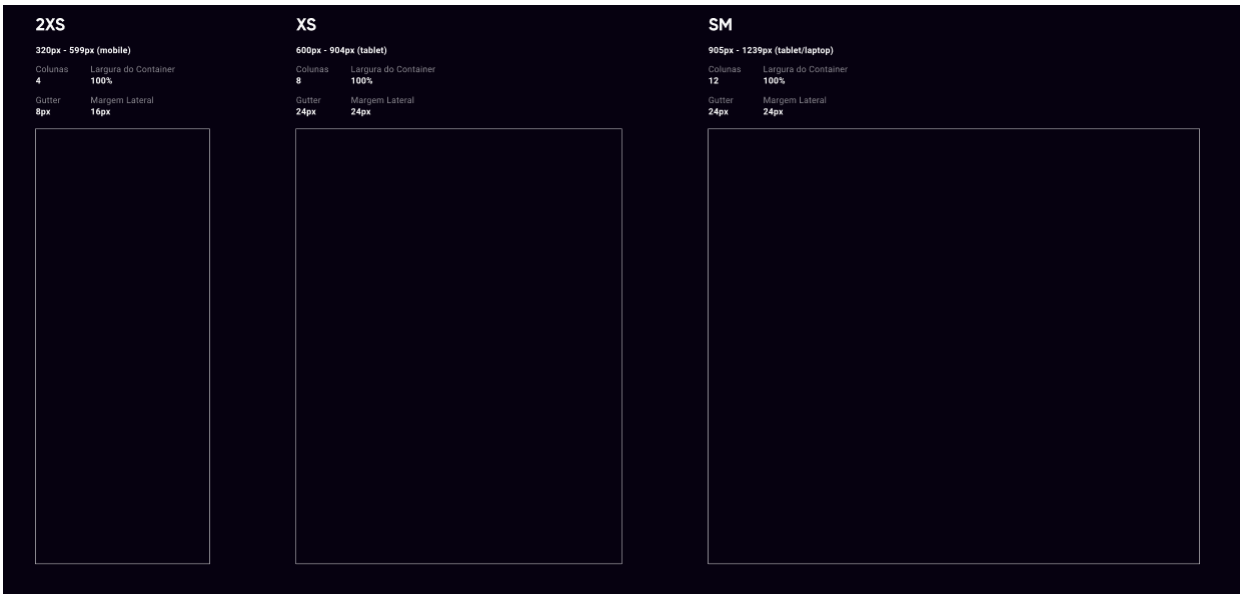
Figura 3.5 – Layout de Breakpoint.



Fonte: Ioasys

Foi definido também os *grids* (grades), que tratam-se de componentes com uma abstração um pouco diferente porém, muito conveniente, pois já trazem uma área toda configurada já com espaços e colunas pré-definidas para colocar seus componentes e não precisar preocupar muito com a distribuição deles. Para ilustrar, na Figura 3.6 há a definição de alguns *grids*. Pensando em um *grind* como um componente, temos um código limpo e uma usabilidade eficiente, pois basta abrir a *tag* do *grid* desejado e colocar todos os componentes a serem utilizados, seguindo a funcionalidade de uma *tag div*.

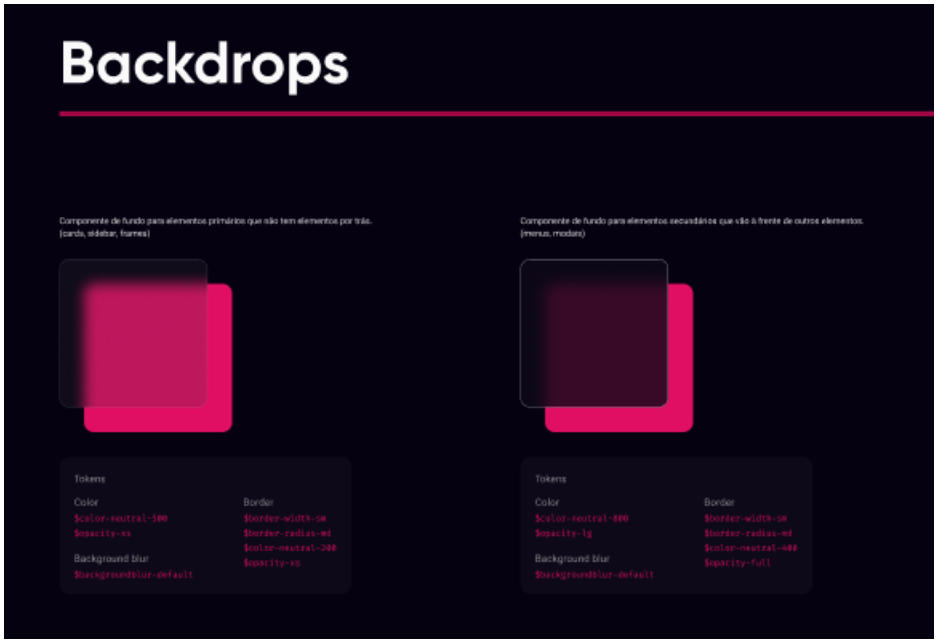
Figura 3.6 – Layout de *Grid*.



Fonte: Ioasys

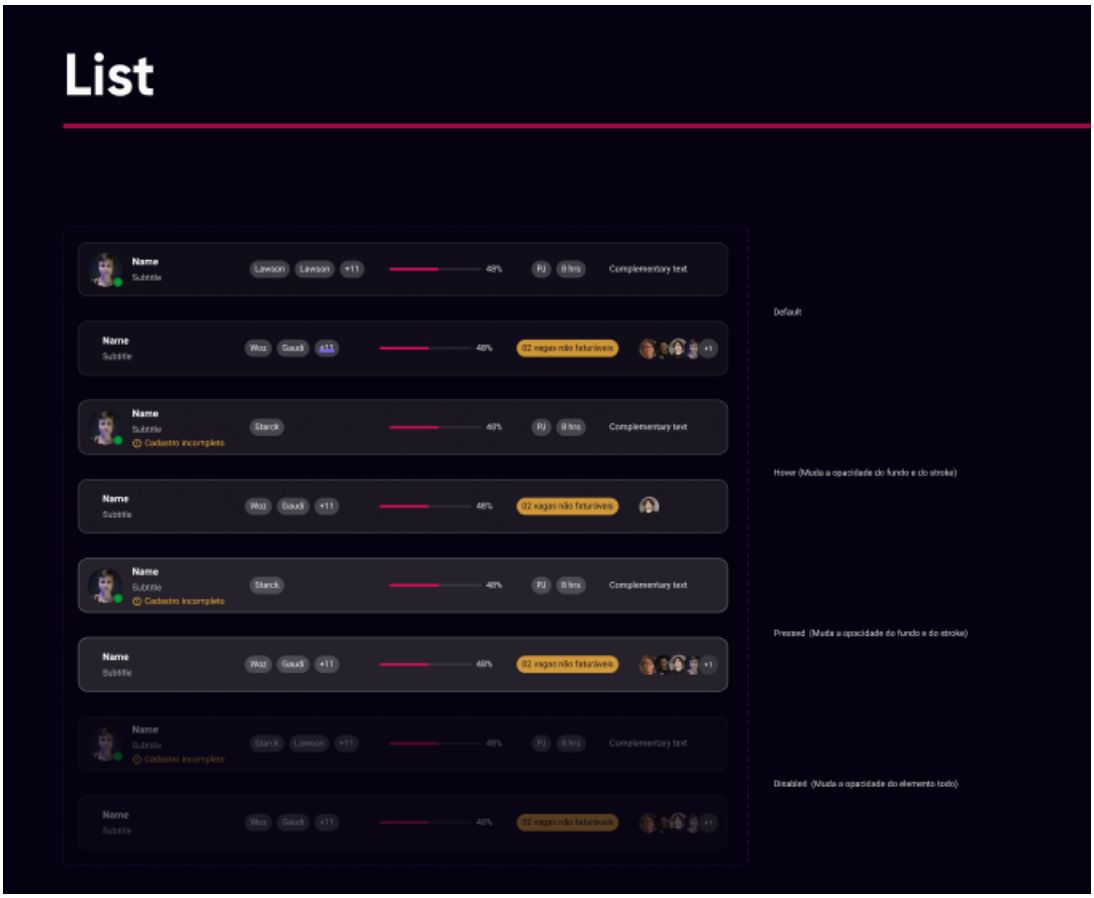
Ainda no *Layout*, temos mais duas definições, sendo elas de *Backdrops* (Cenários) que são as regras para um componente que venha sobre outro ou alguma coisa, como exemplo a opacidade ou *blur*, como mostrado na Figura 3.7. E, por fim, temos as definições do *List* (Lista), que são as características que um *List* deve ter dentro de algum componente, como mostrado na Figura 3.8.

Figura 3.7 – Layout de backdrops.



Fonte: Ioasys

Figura 3.8 – Layout de List.



Fonte: Ioasys

3.1.3 Tipografia

Essa parte também traz uma abstração um pouco diferente, na qual temos toda a tipografia pensada como componentes, ou seja, na implementação de um componente ao invés de ser utilizadas as *tags* de texto padrões do HTML, como mostrado no capítulo 2, utilizamos as "*tags*" de tipografia já com as personalizações aplicadas e padronizadas. Conforme a Figura 3.9 temos a utilização dos *tokens* para padronizar toda a tipografia do Legacy.

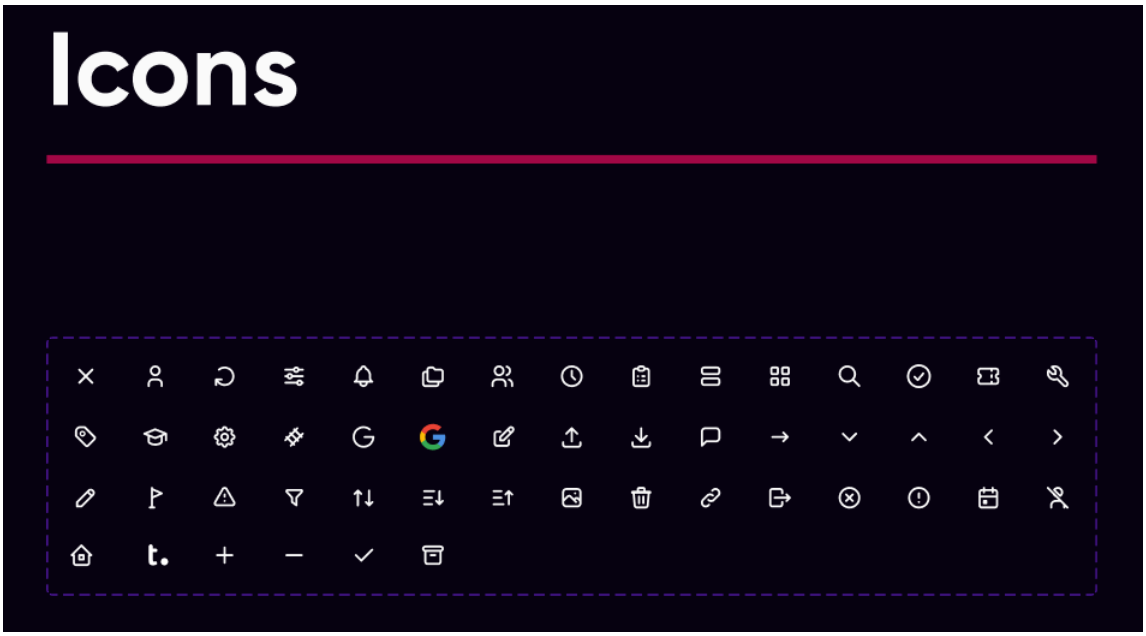
Figura 3.9 – Exemplo de usabilidade dos tokens.

Fonte: Ioasys

3.1.4 Ícones

Em finalidade, temos a última definição de padronização que se configuram como ícones e, para manter tal padrão, todos os ícones já são inseridos no projeto e, com isso, facilita a utilização. Nesse sentido, a mesma abstração do componente de tipografia tem um componente *icon* em que há acesso a todos os ícones da imagem 3.10 alterando apenas o seu parâmetro, bem como é possível alterar o seu tamanho passando de um parâmetro para o outro.

Figura 3.10 – Ícones utilizados no Legacy.



Fonte: Ioasys

3.2 Desenvolvimento do Legacy

O *Legacy* foi implementado com a biblioteca *React* e a linguagem de programação *TypeScript*. Foi decidido pela utilização de *React* devido a sua a ideia de programação voltada a componentização, que é compatível com o conceito de um *design system*.

O *Legacy* está disponível como uma biblioteca NPM. O NPM (*Node Package Manager*) foi criado como um projeto de código aberto para ajudar os desenvolvedores de *JavaScript* a compartilhar facilmente módulos de código empacotados como o *Legacy* (NPM, INC, 2022). Este formato garante simplicidade na sua importação, através do comando *NPM i @Ioasys-Legacy/core*. Após a importação do *Legacy*, o *design system* está pronto para ser utilizado em um projeto, apenas sendo necessário incluir os componentes desejados nos documentos.

Os componentes do Legacy são organizados nas seguintes categorias: *Alerts*, *Buttons*, *Controls*, *Inputs*, *Navigation* e *Tags*. As categorias que foram criadas para o *Legacy* estão detalhadas na tabela a seguir.

Categorias	Descrição
Alerts	Foi classificado como *Alerts* todos os componentes que tenham como função principal passar alguma informação por meio de notificações.
Buttons	Essa categoria foi criada pois implementamos diversos tipos de botões que variam entre tamanho e cor, como também podem ter ícones,conter *labels* ou não, dentre outras alterações.
Controls	Na categoria de *Controls* foram agrupados todos os componentes que tem como função principal interagir e/ou controlar outros componentes, como por exemplo uma *scrollbar* que é utilizada para controlar uma página e também um outro componente interno.
Inputs	A categoria de *Inputs* segue bastante a ideia dos *Buttons*, pois nela implementamos diversos tipos de componentes *inputs* com diversas customizações, como: *text area*, *upload file*, *input select*, etc.
Navigation	Nessa categoria, foram agrupados os componentes utilizados para navegação entre páginas e menus.
Tags	Em *Tags* ocorre a junção dos componentes que são utilizados para rotular determinada página ou outro componente.

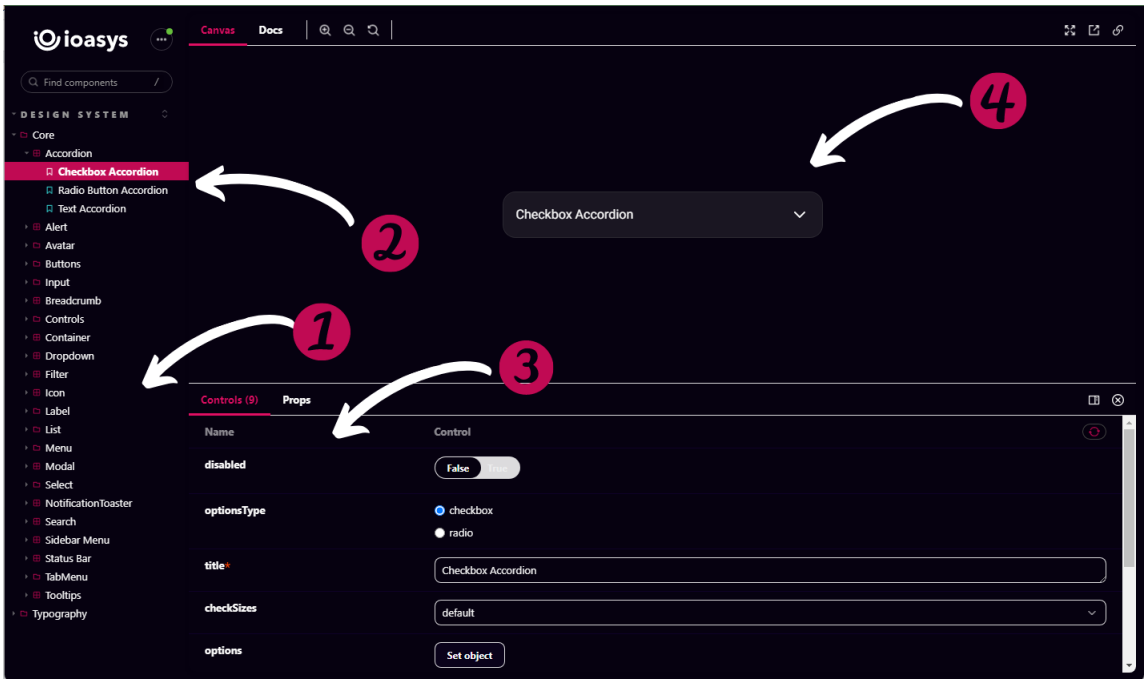
O *Legacy* foi implementado na forma de um *Story Book*, conforme descrito na subseção a seguir.

3.2.1 O Storybook do Legacy

Na Figura 3.11, é possível ver a tela principal do *Storybook* do *Legacy* que, por mais que seja uma tela organizada e intuitiva, está demarcada com setas numeradas para facilitar a explicação de algumas áreas importantes que merecem serem citadas. A primeira seta demarca a listagem de todos os componentes já implementados e que fica separada por 2 tipos: primeiramente os que seguem com um ícone de pasta, que demonstram uma categoria como já citada no trabalho em capítulos anteriores. Dentro dessa pasta há componentes que fazem parte da mesma categoria, algo que mantém o projeto mais organizado. Em sequência, o segundo tipo são componentes livres de categorias, ou seja, são componentes mais específicos que não encaixam nas área definidas. Nessa segunda seta, é perceptível a expansão de um componente que tenha mais de uma variante, como nesse exemplo em que o *accordion* tem mais de uma variação e que optamos por mostrar todas separadamente, assim facilitando para os desenvolvedores que forem utilizar o *Legacy* a terem uma visibilidade maior de todas as variantes do componente que deseja utilizar. Seguindo para a próxima seta em ordem numérica, temos a parte

de controles do componente que é muito utilizada para testes na hora da implementação, como também para desenvolvedores que forem utilizar conseguirem ver todos os parâmetros e ou variações que aquele componente aceita por padrão. E, por último, a quarta seta trata-se da área que será renderizado o componente para visualização e, com tudo feito em tempo real, qualquer alteração nos controles do componente surtirá efeito imediato na visualização do mesmo.

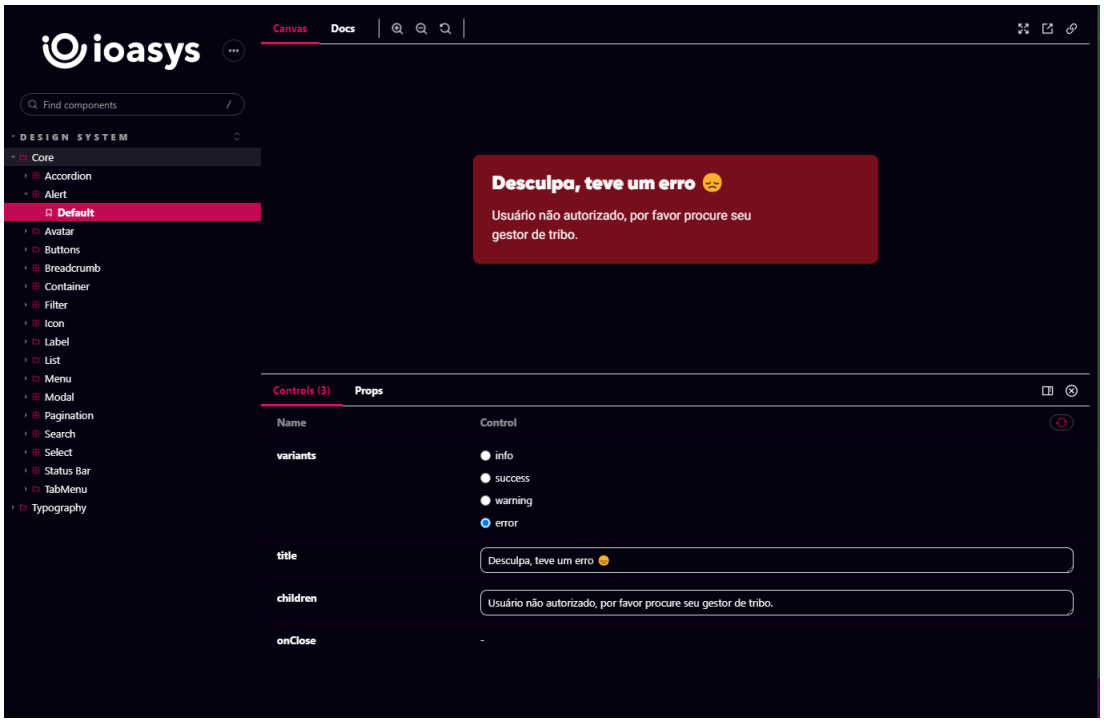
Figura 3.11 – Tela principal do Storybook do Legacy.



Fonte: Ioasys

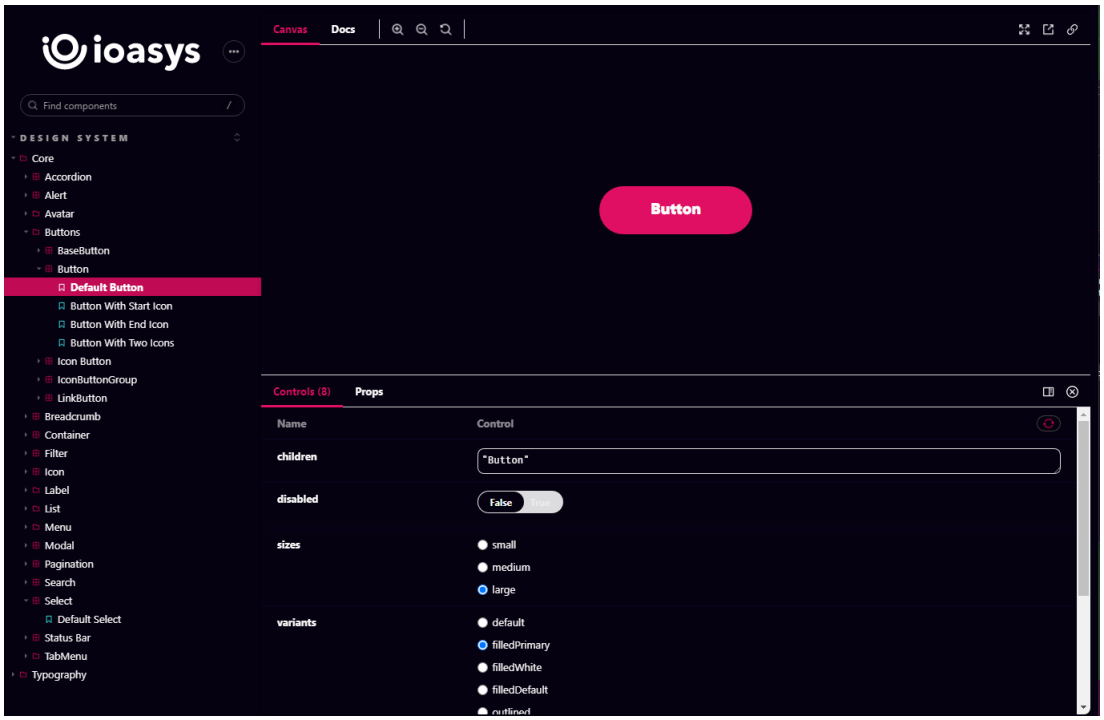
Exemplos de componentes documentados no *StoryBook* são apresentados nas Figuras 3.12, 3.13 e 3.14, respectivamente ilustrando Componente de alerta, Componente de botão e Componente *select*.

Figura 3.12 – Componente de alerta.



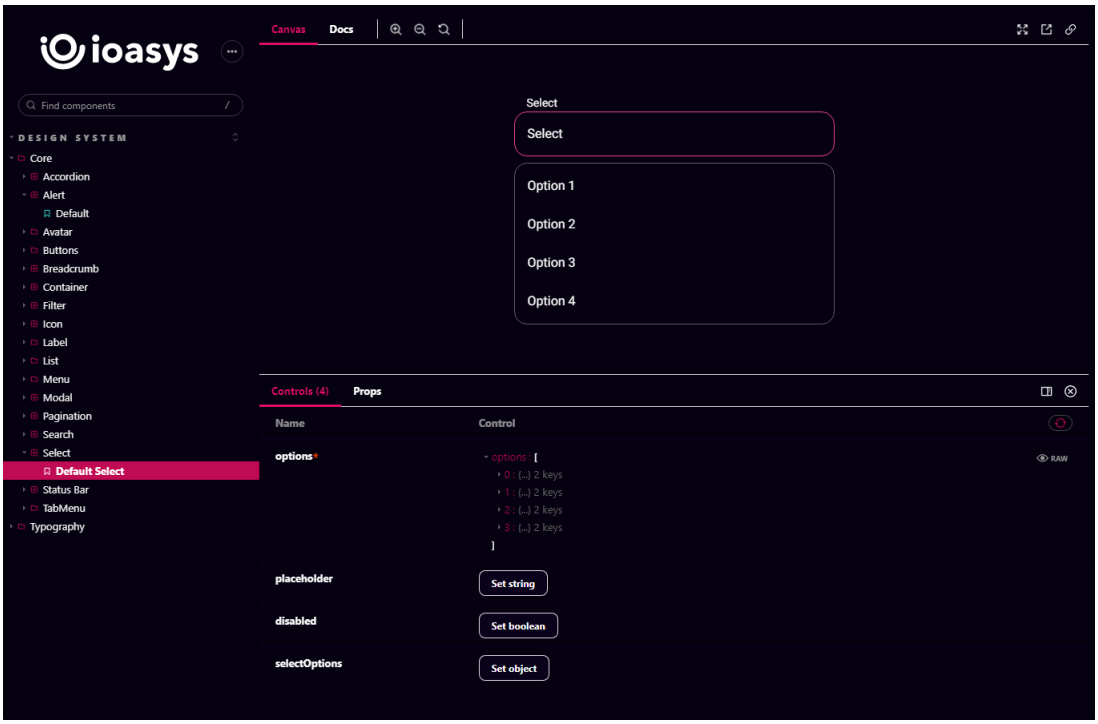
Fonte: Ioasys

Figura 3.13 – Componente de botão.



Fonte: Ioasys

Figura 3.14 – Componente de *select*.

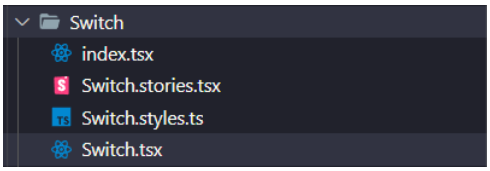


Fonte: Ioasys

3.3 Desenvolvimento de um componente na prática

Neste capítulo será demonstrado sucintamente como foi desenvolvido um componente na prática, com auxílio do código fonte desenvolvido durante o período de estágio. Na figura 3.15 é demonstrado como é a estrutura de organização dos componentes do legacy, ele é composta por uma pasta que leva o nome do componente que dentro contem 5 arquivos que são, respectivamente: um arquivo responsável pela exportação do componente, o segundo é responsável pela documentação do componente no Storybook, o arquivo seguinte é a estilização do componente, o próximo é o arquivo principal onde é desenvolvido a lógica do mesmo e, por fim, o último arquivo é responsável pelos testes do componente.

Figura 3.15 – Estrutura de pastas do componente de switch.



Fonte: Ioasys

A implementação de um componente por padrão tem o costume de ser iniciada pela sua estilização, pois como é utilizada Styled component a implementação iniciada pela estilização é mais intuitiva. Como mostrado na Figura 3.16, a estilização do componente é feita utilizando as *tags* do HTML, porém estilizadas para aquele componente.

Figura 3.16 – Styled component na prática.

```
export const Wrapper = styled.div`
  position: relative;
  max-width: 100%;
  max-height: 100%;
`;
```

Fonte: Ioasys

Indo para o arquivo principal, é nele que é feita toda a lógica de programação do componente, visando sempre a sua reutilização e deixando-o mais modificável possível. Na Figura 3.17, podemos ver que é uma implementação bastante simples e muito comum em componentes React a serem reutilizados, porém neste caso são exportados para serem reutilizados em outros projetos. Na linha 23 podemos ver como um Styled component é utilizado na prática e segue tendo seu papel de div, porém já estilizado e com nome mais fácil de referenciar.

Figura 3.17 – Código de implementação do componente de *switch*.

```tsx
import { ChangeEventHandler } from 'react';
import * as S from './Switch.styles';

export type SwitchProps = {
  /**
   * Switch disabled;
   */
  disabled?: boolean;

  /**
   * Input onChange;
   */
  onChange?: ChangeEventHandler<HTMLInputElement>;

  /**
   *
    Switch value props;
   */
  value?: boolean;
};

const Switch = ({ disabled = false, onChange, value = false }: SwitchProps) => (
  <S.Wrapper>
    <S.SwitchCheck
      type="checkbox"
      id="checkbox"
      disabled={disabled}
      onChange={onChange}
      defaultChecked={value}
    />
    <S.SwitchSlider htmlFor="checkbox" disabled={disabled} role="switch" />
  </S.Wrapper>
);

export default Switch;
```

Fonte: Ioasys

Com a estrutura do componente já implementada, é possível desenvolver sua documentação no Storybook para ir testando conforme anda o desenvolvimento, como também testar suas funcionalidades. Na figura 3.18, temos a implementação do Storybook do componente switch, um componente bastante simples e sem muitas funções. Por esse motivo temos um Storybook mais enxuto, onde as únicas variáveis modificáveis são *"disable"* e *"value"*. Desse modo, *"disabled"* é destinado para exibir o componente com uma estilização diferente quando ele estiver desativado, e *"value"*, como o próprio nome diz, é a variável que guarda seu valor que, nesse caso, é verdadeiro ou falso.

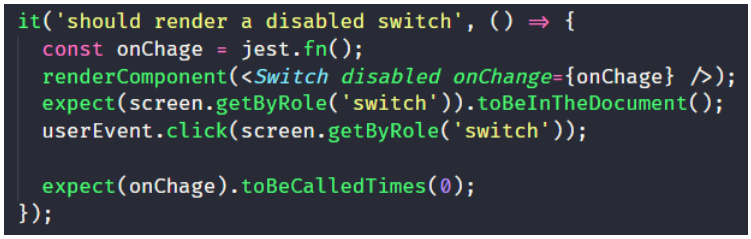
Figura 3.18 – Código do storybook do componente de switch.

```
import { Meta, Story } from '@storybook/react/types-6-0';
import Switch, { SwitchProps } from './Switch';

export default {
  component: Switch,
  title: 'Design System/Core/Controls/Switch',
} as Meta;

const Template: Story<SwitchProps> = (args) => <Switch {...args} />;

export const DefaultSelect = Template.bind({});

DefaultSelect.args = {
  disabled: false,
  value: false,
};
```

Fonte: Ioasys

Por fim, quando o componente é finalizado, são feitos testes unitários e para cobrir o maior numero de casos, são feitos diversos teste, e com isso gerar componentes menos propícios a falhas. Na Figura 3.19, temos o teste responsável por verificar se a estilização do switch desativada era aplicada quando se renderizava essa versão do mesmo.

Figura 3.19 – Código do teste unitário do componente de switch.



```
it('should render a disabled switch', () => {
  const onChage = jest.fn();
  renderComponent(<Switch disabled onChange={onChage} />);
  expect(screen.getByRole('switch')).toBeInTheDocument();
  userEvent.click(screen.getByRole('switch'));

  expect(onChage).toBeCalledTimes(0);
});
```

Fonte: Ioasys

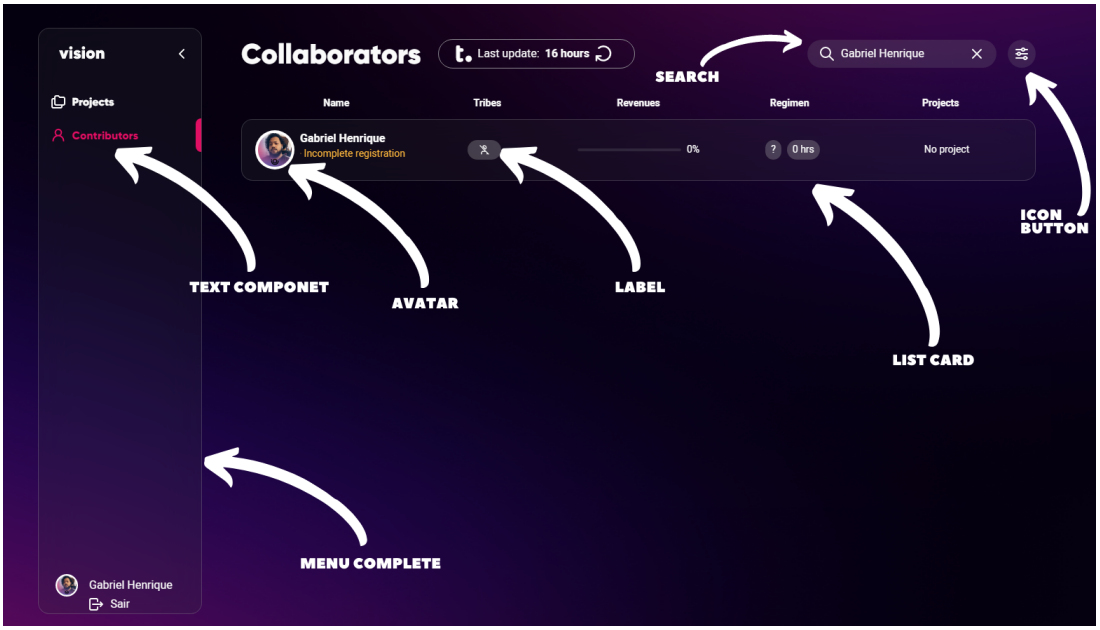
Quando o estagiário começou a atuar no projeto ele não teve contato com a parte de coleta de requisitos e criação do design dos componentes, apenas a parte de implementação dos mesmos nos quais foram citados neste capítulo.

3.4 Aplicando o Legacy no desenvolvimento do Vision

Durante o tempo em que o estagiário atuou na empresa Ioasys, ele não só participou da implementação do design system como também pode utilizá-lo no primeiro projeto interno que fez uso

do *Legacy*: o *Vision*. O *Vision* nasceu a partir da dor que a empresa tinha de gerir seus colaboradores e clientes, pois tudo isso e mais hoje é feito através de planilhas muito grandes que são acessadas a todo momento por muitas pessoas. Então o *Vision* foi pensando para isso, para substituir todas essas planilhas por um projeto completo e com uma interface agradável com chances de erros bem menores. Atualmente, o *Vision* está em desenvolvimento ainda porém já tem algumas telas com seus design projetados e telas já implementadas que serão mostradas neste capítulo. O intuito do Legacy foi visto quando a implantação do *Vision* começou fazendo com que a implementação do mesmo fosse simples e intuitiva. Na Figura 3.20 temos a demonstração de como ficou uma tela construída inteiramente com os componentes do Legacy, as setas demonstram algum dos componentes que foram utilizados nessa tela, desde componentes maiores como o *menu complete* que é uma *sidebar* com algumas informações, também com componentes simples e texto utilizado em toda a pagina, assim como também é um elemento do *Legacy*.

Figura 3.20 – Tela do vision construída inteira com componentes do Legacy.

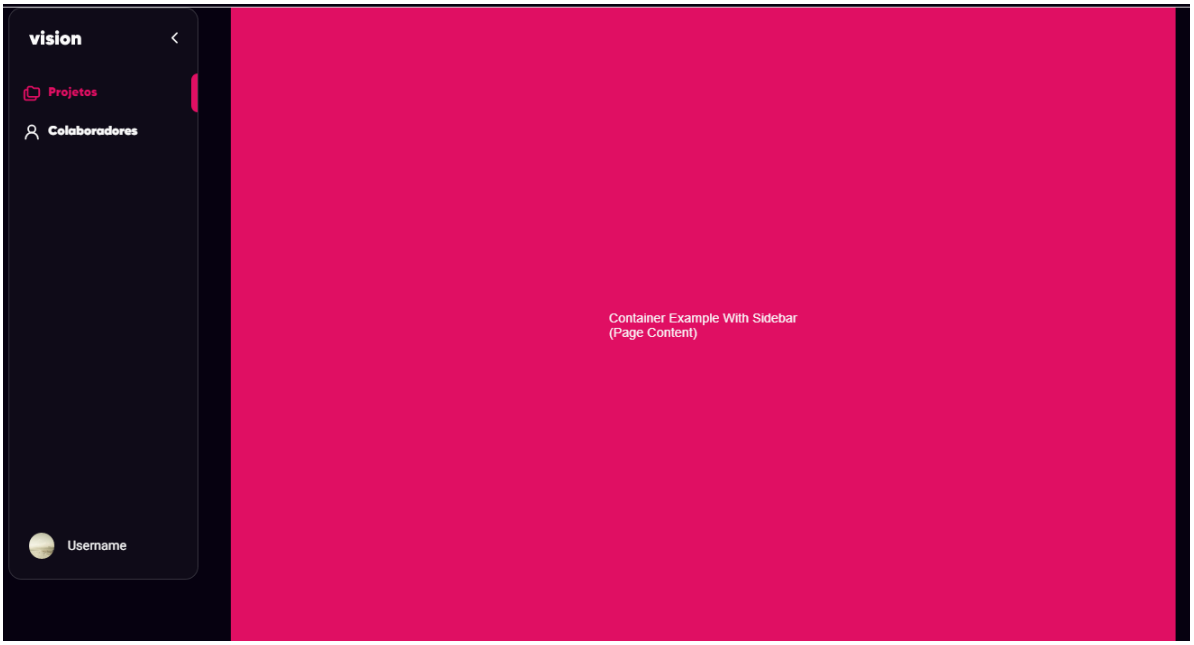


Fonte: Ioasys

Um componente interessante e que possui uma abstração bem diferente é o componente de *container*, ele é um componente que delimita uma área que o desenvolvedor deve construir sua tela

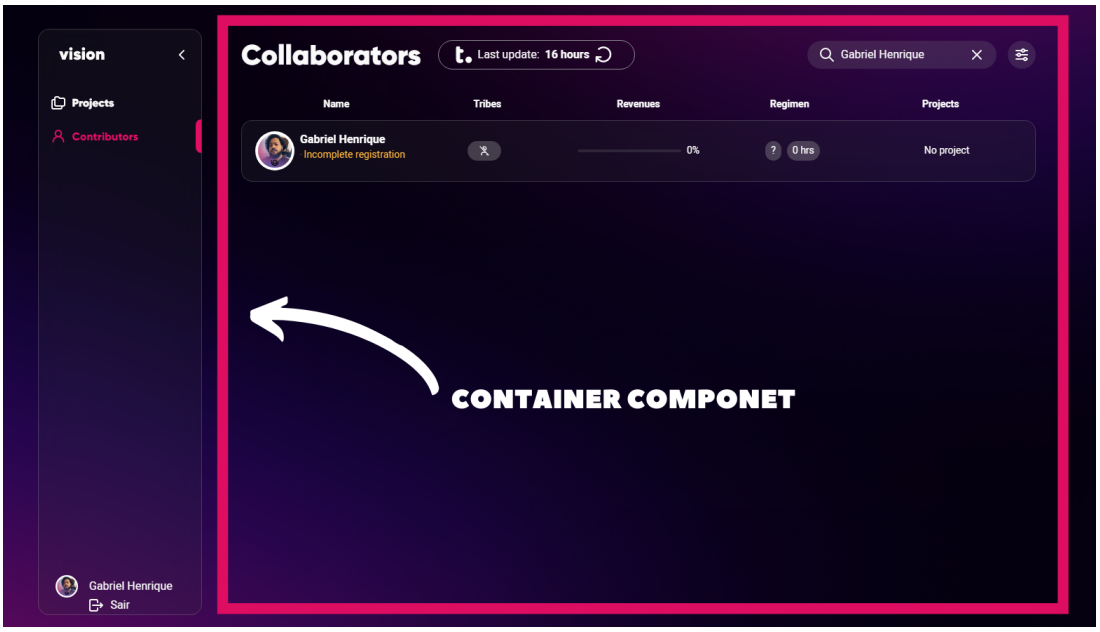
como se ele fosse uma tag div do html. Porém, como mostrado na Figura 3.21 temos o *container* já com a *sidebar*, o que facilita muito quando se pensa na construção do *Vision* já que todas as suas telas levam a esse menu lateral e, assim, facilitando no desenvolvimento de novas telas já com um espaço pré-definido, o que poupa bastante tempo e esforço do desenvolvedor ao ficar estipulado regras para aplicação para manter os componentes em seus devidos lugares. Desse modo, na Figura 3.22 tem-se esse componente utilizado na prática, porém, como ele não é visualizável, foi demarcado com um retângulo em toda sua área de atuação.

Figura 3.21 – Componente de container do Legacy.



Fonte: Ioasys

Figura 3.22 – Aplicação do componente de container no vision.



Fonte: Ioasys

3.5 Considerações Finais

Este capítulo apresentou as atividades desenvolvidas pelo estagiário relacionadas à construção de um *design system* (o Legacy) e sua aplicação no desenvolvimento de um produto interno da organização (o Vision). Percebeu-se que o uso do *Legacy* permitiu que o desenvolvimento do *Vision* fosse mais simples visando a reutilização de componentes ao em vez de começar do zero, como também foi ditado os padrões de design que os projetos internos da Ioasys terão daqui pra frente.

4 CONCLUSÃO

As atividades desenvolvidas e os conhecimentos adquiridos e aprimorados durante o período no qual se decorreu o processo de estágio foram de grande valia para a formação de pilares de desenvolvimento e aprimoramento tanto pessoal quanto profissional do tutorado, processo ao qual todos os envolvidos que tiveram contato direto ou indireto corroboraram para tal desenvolvimento e aperfeiçoamento do estagiário. Essas pessoas, desde o mais breve contato ao mais continuo acompanhamento no processo de estágio contribuíram de diversas maneiras e com o mínimo dos gestos passaram valiosos ensinamentos. No período em que se decorreu o processo de estágio foi introduzido no ambiente do tutorado diversas tecnologias, muitas das quais são muito recentes e de grande relevância no cenário atual do mercado de trabalho. Mesmo sendo tecnologias evoluídas ou distintas daquelas introduzidas, desenvolvidas e aplicadas durante o período de graduação, foi pertinente para a aplicação de diversos conhecimentos adquiridos e construídos pelo discente no decorrer de sua formação. Conhecimentos estes que se estendem desde os princípios básicos da programação como a lógica e a introdução à algoritmos até boas práticas para desenvolvimento de um software limpo, seguro e consistente, assim visto em disciplinas como engenharia de software, introdução à computação, paradigmas de linguagens de programação e muitas outras.

Um dos conhecimentos básicos de um bom profissional que foi muito exercitado pelo tutorado durante o processo de estágio foi a cooperação e trabalho em equipe, mesmo tendo sido introduzido em dinâmicas cooperativas durante o período de graduação, por diversas disciplinas e professores, foi na divisão de tarefas e equipes na dinâmica corriqueira de uma empresa de desenvolvimento de software que fundamentou-se esses conhecimentos e concretizou essa maturidade de um bom profissional de desenvolvimento de software. O contato direto com o mercado de trabalho e a exposição à volatilidade do mundo corporativo isenta o conforto do ambiente controlado que é a sala de aula e é nesse momento que o discente, frente ao mercado de trabalho, consegue abranger-se do estado de aluno para o estado de um profissional. É de suma relevância salientar que durante o período de graduação o discente cursou uma disciplina denominada de Modelagem e Implementação de Software, na qual o principal objetivo almejado era simular funcionamento do dia a dia de uma empresa de desenvolvimento de software, desde a divisão cooperativa de trabalho em equipe até a interação e negociação com cliente

e potenciais consumidores da tecnologia desenvolvida. Experiencia essa que foi imprescindível para preparar o discente às expectativas do mundo corporativo, permitindo-o adaptar-se e compreender de maneira muito satisfatória o universo que o esperava à frente.

REFERÊNCIAS

BIERMAN, G.; ABADI, M.; TORGERSEN, M. Understanding typescript. In: SPRINGER. **European Conference on Object-Oriented Programming**. [S.l.], 2014. p. 257–281.

CARVALHO, B. V. d.; MELLO, C. H. P. Aplicação do método ágil scrum no desenvolvimento de produtos de software em uma pequena empresa de base tecnológica. **Gestão & Produção**, SciELO Brasil, v. 19, n. 3, p. 557–573, 2012.

GODBOLT, M. **Frontend architecture for design systems: a modern blueprint for scalable and sustainable websites**. [S.l.]: "O'Reilly Media, Inc.", 2016.

JIM HIGHSMITH. **History: The Agile Manifesto**. Lavras, 2001. Disponível em: <https://agilemanifesto.org/history.html>. Acesso em: 18 abr. 2022.

MEIUCA. **Jota DS**. Lavras, 2022. Disponível em: <https://zeroheight.jota.meiuca.co/561dd07ed/p/9467c2-jota-ds>. Acesso em: 6 abr. 2022.

META PLATFORMS. **React - Uma biblioteca JavaScript para criar interfaces de usuário.** Lavras, 2022. Disponível em: <https://pt-br.reactjs.org/>. Acesso em: 23 fev. 2022.

MILETTO, E. M.; BERTAGNOLLI, S. de C. **Desenvolvimento de Software II: Introdução ao Desenvolvimento Web com HTML, CSS, JavaScript e PHP-Eixo: Informação e Comunicação-Série Tekne**. [S.l.]: Bookman Editora, 2014.

MOZILLA. **CSS**. Lavras, 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/CSS>. Acesso em: 21 fev. 2022.

MOZILLA. **<h1>–<h6>: Os elementos HTML de cabeçalho da seção**. Lavras, 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/Heading_Elements>. Acesso em: 18 fev. 2022.

MOZILLA. **HTML: Linguagem de Marcação de Hipertexto**. Lavras, 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML>. Acesso em: 17 fev. 2022.

MOZILLA. **JavaScript**. Lavras, 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>. Acesso em: 17 fev. 2022.

NPM, INC. **About npm**. Lavras, 2022. Disponível em: <https://www.npmjs.com/about>. Acesso em: 6 abr. 2022.

SCHWABER, K.; SUTHERLAND, J. The 2020 scrum guide™. 2020. **URL: https://scrumguides. org/scrum-guide. html (hämtad 2021-03-03)**, 2020.

THE MIT LICENSE. **Jota DS**. Lavras, 2022. Disponível em: <https://storybook.js.org/>. Acesso em: 18 abr. 2022.

VESSELOV, S.; DAVIS, T. **Building Design Systems: Unify User Experiences through a Shared Design Language**. [S.l.]: Apress, 2019.