UNIVERSIDADE FEDERAL DO AMAZONAS - UFAM

FACULDADE DE TECNOLOGIA - FT

BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

Proto2Code: Uma ferramenta para geração de código front-end a partir de protótipos de telas

Mailton Viana Farias

Manaus - AM

Abril de 2022

Mailton Viana Farias

Proto2Code: Uma ferramenta para geração de código front-end a partir de protótipos de telas

Monografia de Graduação apresentada à Coordenação de Engenharia da Computação, UFAM, da Universidade Federal do Amazonas, como parte dos requisitos necessários à obtenção do título de Engenheiro da Computação.

Orientador:

Bruno Freitas Gadelha, Dr.

Coorientadora:

Gretchen Torres de Macedo, Msc.

Universidade Federal do Amazonas - UFAM

Faculdade de Tecnologia - FT

Manaus - AM

Abril de 2022

Ficha Catalográfica

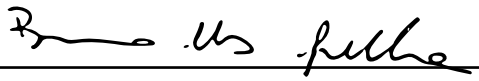
Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Farias, Mailton Viana
F224p Proto2Code: uma ferramenta para geração de código front-end a partir de protótipos de telas / Mailton Viana Farias . 2022
61 f.: il. color; 31 cm.

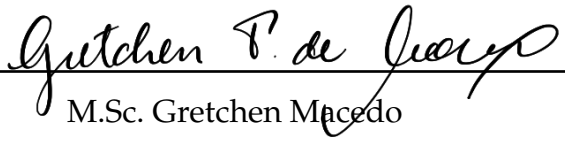
Orientador: Bruno Freitas Gadelha
Coorientadora: Gretchen Torres de Macedo
TCC de Graduação (Engenharia da Computação) - Universidade Federal do Amazonas.

1. Engenharia de software com reuso. 2. Prototipação. 3. Geração de código. 4. Front-end. I. Gadelha, Bruno Freitas. II. Universidade Federal do Amazonas III. Título

Monografia de Graduação submetida sob o título *PROTO2CODE: UMA FERRAMENTA PARA GERAÇÃO DE CÓDIGO FRONT-END A PARTIR DE PROTÓTIPOS DE TELAS* por Mailton Viana Farias e aceita pelo Instituto de Computação da Universidade Federal do Amazonas, sendo aprovada por todos os membros da banca examinadora abaixo especificada:

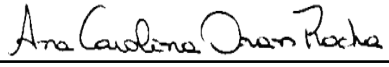


Dr. Bruno Gadelha
Orientador:
Instituto de Computação
Universidade Federal do Amazonas



M.Sc. Gretchen Macedo
Coorientador(a)
Instituto de Computação
Universidade Federal do Amazonas

Dr. David Fernandes
Instituto de Computação
Universidade Federal do Amazonas



Dra. Ana Carolina Oran
Instituto de Computação
Universidade Federal do Amazonas

Manaus - AM, 19 de abril de 2022

Dedico este trabalho a quem me deu a oportunidade de embarcar nesse projeto e colaborou diretamente para a conclusão dele: meu orientador Bruno Gadelha e minha coorientadora Gretchen Macedo, sem os quais eu não teria tido esses resultados e dado esse último passo para a conclusão da minha graduação.

Agradecimentos

Em primeiro lugar, a Deus, pelo dom da vida e que fez com que meus objetivos fossem alcançados durante todos os meus anos de estudos, mantendo-me saudável e perseverante. Aos meus pais, Mariete Libório e Anailton Farias que investiram na minha educação, contribuíram para que eu chegasse até o ensino superior, me sustentaram durante todo esse tempo e nunca me deixaram faltar as coisas básicas para eu estar na Universidade.

Estendo esse agradecimento à minha namorada, pelo apoio e pela admiração, à minha irmã, aos familiares e aos amigos que torcem para o meu sucesso e se importam com a minha vida acadêmica e profissional e a todos que contribuíram direta e indiretamente para que eu chegasse até o final dessa graduação, seja aos poucos amigos de curso ou amigos fora da Universidade.

Além disso, agradeço à Universidade Federal do Amazonas por toda a estrutura e pelas oportunidades, sempre defenderei a UFAM e ao ensino público de qualidade. Agradeço também ao corpo docente dos cursos de Computação, em especial aos meus orientadores Bruno e Gretchen que se dispuseram a me orientar sem me conhecer, agradeço pela simpatia, pelo comprometimento e pela honestidade. Meu muito obrigado a todos.

Proto2Code: Uma ferramenta para geração de código front-end a partir de protótipos de telas

Autor: Mailton Viana Farias

Orientador: Bruno Freitas Gadelha, Dr.

Resumo

Desenvolvedores *Front-End* hoje em dia dispõem de vários *frameworks* e *softwares* que os auxiliam no desenvolvimento de suas aplicações. Durante o desenvolvimento de *software*, algumas atividades comuns requerem tempo e esforço dos desenvolvedores como por exemplo criação de cadastros básicos com rotinas de criação, leitura, alteração e exclusão de dado. Mais especificamente, no caso de desenvolvimento de interfaces, muito tempo é investido na implementação *front-end* desses sistemas. Com isso, ferramentas podem ser utilizadas para diminuir esse esfoço de desenvolvimento ou para os desenvolvedores. Este trabalho apresenta a ferramenta Proto2Code, que visa acelerar o processo de desenvolvimento de aplicações através de geração de código *front-end* (HTML e CSS) a partir de protótipos de interface. Para isso, foi adotada uma metodologia que é dividida em seis etapas para a construção de um gerador de código. Para validar essa versão da ferramenta, realizou-se um estudo piloto com dois desenvolvedores webs que participaram, em duas etapas, no uso da ferramenta. Como resultado, verificou-se que a ferramenta auxilia desenvolvedores menos experientes que utilizam ferramentas mais tradicionais de desenvolvimento, dependendo também, do tipo de protótipo utilizado.

Palavras-chave: Engenharia de Software com reuso, prototipação, geração de código, front-end.

Proto2Code: Uma ferramenta para geração de código front-end a partir de protótipos de telas

Autor: Mailton Viana Farias

Orientador: Bruno Freitas Gadelha, Dr.

Abstract

Front-End developers nowadays have several frameworks and software that help them develop their applications. During software development, some common activities require time and effort from the developers, such as the creation of basic registrations with routines for creating, reading, changing, and deleting data. More specifically, in the case of interface development, a lot of time is invested in the front-end implementation of these systems. Therefore, tools can be used to reduce this effort for developers. This work presents the Proto2Code tool, which can speed up the development process by generating front-end code (HTML and CSS) from interface prototypes. For this, a methodology was adopted which is divided into six steps for the construction of a code generator. To validate this version of the tool, a pilot study was conducted with two web developers who participated, in two stages, in the use of the tool. As a result, we found that the tool helps less experienced developers who use more traditional development tools, depending also on the type of prototype used.

Keywords: Software engineering with reuse, prototyping, front-end, code generation.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1

INTRODUÇÃO

O processo de desenvolvimento de *software* mudou consideravelmente nas últimas décadas para atender às necessidades do mercado competitivo. Assim como a indústria de *software* aumenta a sua capacidade de produção, também aumenta a demanda por sistemas mais baratos, eficientes, confiáveis e que podem atuar sobre fortes restrições, não apenas de custo, mas também de tempo (ALÉSSIO; SABADIN; ZANCHETT, 2017).

Equipes de desenvolvimento de *software* muitas vezes fazem protótipos de tela para discutir com os clientes, por ser um meio rápido e bastante eficiente para validar uma ideia. Além de melhorar a captação e o processo de elicitação de requisitos, por meio dos protótipos apresenta-se para o cliente o entendimento dos desenvolvedores de como o problema será resolvido. A prototipação é uma etapa, embora opcional, importantíssima no desenvolvimento de *software*, pois a sua execução influencia diretamente a produtividade de uma equipe e os valores entregues ao cliente por uma empresa ou para quem está desenvolvendo (CroSoften, 2021).

Porém, muitas vezes o protótipo é descartado e o desenvolvimento do sistema segue seu fluxo tradicional (SOMMERVILLE, 2011). Na indústria, com tudo que já existe na internet hoje em dia, essa maneira tradicional de desenvolver sistemas a partir do zero já não atende a estas demandas. A indústria de *software* tem sofrido algumas modificações para tentar resolver estes problemas: tempo e otimização, pois ainda se leva muito tempo para sair do campo das ideias até chegar no sistema implementado.

Uma outra abordagem que se relaciona a estes problemas de tempo e otimização

de desenvolvimento que será discutida neste trabalho é o reuso. Reusar componentes de *software* significa reduzir custos e tempo no processo de desenvolvimento, além de aumentar a qualidade entre outras diversas vantagens que tornam esse conceito praticável. Reuso explora semelhanças nos requisitos/arquitetura entre aplicações. Visto ainda que o reuso de uma aplicação para originar outra deve possuir requisitos ou arquiteturas semelhantes, pode-se dizer que para este trabalho o reuso de *software* é uma das alternativas (FERREIRA; NAVES, 2011). Então, neste trabalho explora-se o reuso no sentido de geração de código a partir de protótipos de interface de telas no processo de desenvolvimento *front-end*.

1.1 Objetivos

Tendo em vista os pontos destacados acima, a grande procura por sistemas web e o tempo perdido com tarefas rotineiras, como por exemplo a criação de cadastros básicos e a dificuldade de integração entre programadores, *web-designers* e desenvolvedores *frond-end* - o desenvolvedor que "lê" um arquivo de *design* e o transforma em um código funcional -, faz-se necessário ter ferramentas ou artefatos que ajudem esse processo de desenvolvimento desses três profissionais, neste caso em especial o do desenvolvedor *front-end*. Então este trabalho tem como objetivo desenvolver uma ferramenta que acelere o processo de desenvolvimento através de geração de código *front-end* a partir de protótipos de interface, através da geração de código HTML (HiperText Markup Language).

1.2 Objetivos específicos

Os objetivos específicos para a proposta deste trabalho são:

- Analisar e identificar os tipos de arquivos de exportação de ferramentas de prototipação.
- Caracterizar uma lista de componentes do Balsamiq Wireframes que são relevantes

e sua representação de acordo com o tipo do arquivo exportado.

- Realizar um mapeamento entre os elementos e a representação deles em código.
- Desenvolver a ferramenta em uma linguagem de programação com os *inputs* e dados mapeados nos objetivos acima.
- Avaliar a ferramenta desenvolvida.

1.3 Metodologia

Para atingir os objetivos propostos neste trabalho, foi realizada uma revisão da literatura acerca dos principais conceitos envolvidos, como Engenharia de Software com reuso, ferramentas de prototipação e geração automática de código para consolidação do entendimento. Depois verificou-se ferramentas de prototipação possíveis de serem adotadas ao contexto do trabalho e foi escolhida a ferramenta Balsamiq Wireframes (detalhada na seção 2.4.2). Após isso, passou-se para a etapa de desenvolvimento da ferramenta de geração de código.

A Figura 1 abaixo melhor representa as etapas da metodologia deste trabalho:

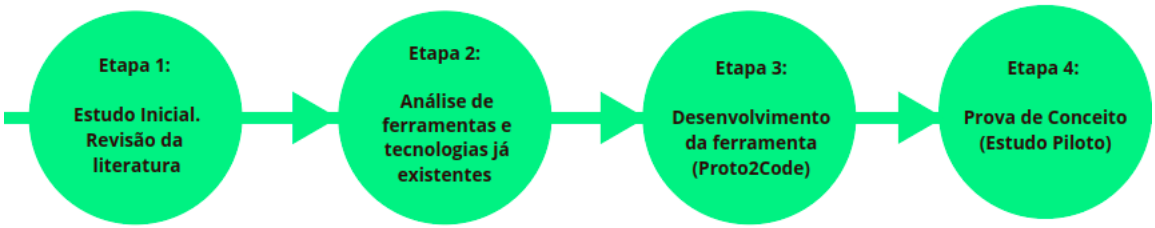


Figura 1 – Metodologia

- A Etapa 1 consiste na revisão da literatura quanto ao estudo de conceitos fundamentais aqui envolvendo Engenharia de Software com reuso, prototipação e ferramentas de prototipação, geração automática de código e desenvolvimento front-end. Além disso, através dessa revisão e estudo da literatura foi feito um levantamento de trabalhos relacionados nos quais são utilizados como referências neste trabalho.

- A Etapa 2 consiste na análise e estudo de ferramentas do mercado sobre desenvolvimento frond-end, onde foram feitos alguns testes com os *frameworks* Vue JS e Rect JS. Além do estudo com HTML, CSS e Bootstrap.

- Já a Etapa 3 é sobre o desenvolvimento da ferramenta descrita neste trabalho. Foi seguido a metodologia citada em Saturno (2007) *apud* Herrington (2003) que descrevem as etapas necessárias para a construção de um gerador de código. Decidiu-se usar como base suas etapas porque são as que mais se encaixam para o desenvolvimento prático da ideia deste trabalho. Essas etapas são passos para a codificação e são descritas no Capítulo 3 deste trabalho.

- Por fim, a Etapa 4 consiste na avaliação desta ferramenta. Ela foi conduzida através de uma prova de conceito [1] que foi realizada em duas etapas: uma etapa consistiu na geração de diversos protótipos e na geração de seus respectivos códigos - a nível de teste para melhoria do *script* da ferramenta -, e na segunda etapa, avaliou-se com dois desenvolvedores de *softwares*. Esta avaliação é melhor descrita no Capítulo 4, onde é explicado com "Estudo Piloto" na seção 4.1.

1.4 Organização do Trabalho

Esta monografia está organizada da seguinte maneira:

- No Capítulo 2, são apresentados os fundamentos teóricos utilizados, como a prototipação e seus tipos e sua importância para a Engenharia de Software. Sobre a Engenharia com Reuso, suas vantagens e desvantagens e comparações e por fim, os trabalhos relacionados a este e as tecnologias utilizadas para o desenvolvimento da ferramenta.

- O Capítulo 3 apresenta sobre a ferramenta Proto2Code em si, como o método de desenvolvimento dela, as etapas, as estratégias usadas e também o modo de usar o *script* como ferramenta desenvolvida.

[1] Prova de Conceito: (Proof of Concept, em inglês, ou simplesmente PoC) é o nome que se dá à demonstração da possibilidade de validação de uma ideia (ou conceito), seja na área de TI ou na de negócios (Blog Na Prática: https://www.napratica.org.br/o-que-e-prova-de-conceito/).

- O Capítulo 4, apresenta os testes e as avaliações realizadas com desenvolvedores reais, onde essa avaliação foi feita em duas partes e é descrita por processos em forma de figura e os resultados também são apresentados.

- O Capítulo 5, por fim, descreve as opiniões do resultados obtidos e as conclusões em geral, bem como as melhorias a realizar e propostas para trabalhos futuros.

- Seguido pelas referências bibliográficas, tem-se o Apêndice que mostra a tabela de mapeamento dos componentes do Balsamiq Wireframes.

2

FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão tratados os elementos teóricos que fundamentam este trabalho, de tal modo que será abordado sobre a prototipação e seus tipos. Sobre a Engenharia de Software com Reuso, suas vantagens e desvantagens, como é estruturada a implementação de engenharia de reuso e seus panoramas.

Por sequência, será discutido sobre a geração automática de código e onde foram aplicados esses fundamentos no desenvolvimento da ferramenta. Por fim, há a comparação com os trabalhos relacionados que foram estudados para embasamento teórico deste trabalho.

2.1 Prototipação

A prototipação pode ser considerada um processo importante no desenvolvimento de *software* porque serve como um primeiro rascunho de um projeto, além de se tratar de uma abordagem voltada à construção de um modelo (protótipo) passível de ser avaliado iterativamente pelos envolvidos com o objetivo de identificar e solidificar as ideias e requisitos do *software* em criação (PRESSMAN; MAXIM, 2016).

O conceito de prototipação é abordado de forma diferente por diferentes autores. Rosemberg (2008) define prototipação como: "Um protótipo é uma representação limitada de um design, a qual pode ser um esboço em papel de uma tela ou conjunto de telas, uma “fotografia” eletrônica, uma simulação em vídeo de uma tarefa, uma maquete tridimensional, de papel ou cartolina, ou um simples conjunto de telas vinculadas por

hyperlinks".

Um outro conceito é trazido por Sommerville (2011): "Um protótipo é uma versão inicial de um sistema de *software*, usado para demonstrar conceitos, experimentar opções de projeto e descobrir mais sobre o problema e suas possíveis soluções", pois um protótipo de *software* pode ser usado em um processo de desenvolvimento de *software* para ajudar a antecipar as mudanças que podem ser requisitadas. A prototipação também é uma parte essencial do processo de projeto da interface de usuário, uma vez que, devido à natureza dinâmica de tais interfaces, descrições textuais e diagramas não são bons o suficiente para expressar seus requisitos. Portanto, a prototipação rápida com envolvimento do usuário final é a única maneira de desenvolver interfaces gráficas de usuário para sistemas de *software* (SOMMERVILLE, 2011).

Já Baumer (1996) conceitua a prototipação como sendo uma abordagem de desenvolvimento usada para melhorar o planejamento e execução de projetos de *softwares* através do desenvolvimento de sistemas de *software* executáveis (protótipos) para fins experimentais. Além de afirmar que a prototipação é um excelente meio para gerar ideias sobre como uma interface de usuário pode ser desenvolvida. Tanto que esta é a razão pela qual a prototipação de interface de usuário é aplicada num número crescente de projetos.

Um protótipo pode ser usado como "meio de comunicação" entre os atores (membros da equipe) de desenvolvimento ou mesmo como meio de testar ideias. Pode-se dizer que quanto mais iterativo for o processo de desenvolvimento do protótipo, melhor será o sistema final (ROSEMBERG et al., 2008). De modo geral, o principal uso é para ajudar os usuários/clientes, junto com os desenvolvedores, a entender e validar os requisitos para o sistema.

Desse modo, atualmente os protótipos têm sido mais utilizados para avaliar características de interfaces (UI), de modo a garantir facilidade de uso do sistema e avaliar se os usuários conseguem atingir seus objetivos. Então, neste trabalho, o conceito de protótipação será usado na acepção fornecida por Sommerville (2011).

Segundo Britto (2011), as técnicas, ou abordagens, são utilizadas em diversos métodos de projeto e avaliação de interfaces, especialmente em testes de usabilidade.

Com isso, temos a classificação dos protótipos.

2.1.1 Classificação de protótipos

Existem na literatura várias classificações para os protótipos, dependendo das características que se quer abordar. Não há uma concordância entre os autores sobre os tipos e as classificações dos protótipos.

Em Lopez (2003), os protótipos são classificados em várias categorias e tipos, como por exemplo:

- Classificação dos protótipos segundo seus objetivos (Prototipação exploratória, prototipação experimental e prototipação evolucionária).
- Classificação dos protótipos segundo o grau de fidelidade (Baixa fidelidade e de alta fidelidade).
- Classificação dos protótipos pelo nível de funcionalidade (Prototipação vertical e prototipação horizontal).
- Classificação dos protótipos pela técnica de construção (Prototipação descartável e prototipação reutilizável).

Em Baumer (1996), é chamado de "abordagens à prototipagem da interface de usuário"e adotam o modelo três "E", muito utilizado na Europa: **e**xploratório, **e**xperimental e **e**volutivo. Quanto à classificação de protótipos, classificam como:

- *Presentation Prototypes* (Protótipos de apresentação);
- *Functional Prototypes* (Protótipos funcionais);
- *Breadboards;*
- *Pilot System* (Sistemas-piloto).

Em Wazlawick (2019), prototipação é um modelo de processo de desenvolvimento de *software* e foca-se na Prototipação Evolucionária, onde o mesmo diz que é

uma técnica que pode ser entendida como um modelo independente ou parte de outro modelo em que protótipos cada vez mais refinados são apresentados ao cliente sobre a evolução de requisitos de forma suave e consistente. O autor distingue duas abordagens de prototipação evolucionária: a) *throw-away* e b) *cornerstone*.

Já segundo Britto (2011), pode-se classificar os protótipos em duas principais categorias, de acordo com suas características citadas abaixo: detalhamento e reuso. Essas duas categorias são separadas por vários níveis. Neste trabalho será adotada a classificação de Britto (2011).

2.1.1.1 Quanto ao nível de detalhamento

O nível de detalhamento se refere à quantidade de detalhes que o protótipo suporta, ou melhor, o quanto ele se assemelha ao produto final, em termos de funcionalidade, interação e interface. Assim, eles podem ser classificados como protótipos de:

a) Baixa fidelidade: é o rascunho da ideia, o que esboça o funcionamento da interface. São aqueles que não se assemelham ao produto final, focam em componentes de interface e interações e tem como principal vantagem a elaboração do protótipo de modo barato. Geralmente é desenvolvido utilizando recursos diferentes dos que serão utilizados para construir a versão final do produto (BRITTO et al., 2011) e (ROSEMBERG et al., 2008).

b) Média fidelidade: é uma solução mais elaborada e considerada intermediária com mais detalhes. Estes geralmente são mais trabalhosos mas são de rápida construção e se utilizam de recursos computacionais que permitem simular o comportamento de interação da interface (BRITTO et al., 2011).

c) Alta fidelidade: o objetivo é, de fato, replicar o comportamento do produto final. Utilizam os mesmos recursos do produto final e resultam em um protótipo que melhor se assemelha à versão final do produto. Estes são geralmente construídos em linguagem de programação e atuam como forma de apresentar o funcionamento

de alguns recursos, incluindo a implementação de algumas rotinas e definições dos aspectos da interface gráfica (BRITTO et al., 2011). Segundo Rosemberg *et. al* (2008), protótipos de alta fidelidade são úteis para demonstrar padrões e guias de estilo.

2.1.1.2 Quanto ao reuso

Os protótipos podem ser reutilizados em ciclos iterativos a fim de refinamento ou serem descartados a cada etapa do processo, isso dependendo do processo de desenvolvimento adotado para o projeto. Assim, podem ser ditos como evolucionários e *throw-away*, que é o mesmo que a prototipação descartável e é como será usado neste trabalho.

a) Protótipos evolucionários: São baseados em iterações e entregas rápidas de protótipos que são refinados a cada iteração até atingir o sistema final. São mais utilizados para sistemas onde a especificação não pode ser desenvolvida à priori. Geralmente, pode não ser possível verificar requisitos caso não haja especificação (BRITTO et al., 2011). Além disso, de acordo com Wazlawick (2019), a prototipação evolucionária permite melhor adaptação às mudanças de requisitos. Porém, diz ter uma tendência a degradar o código devido às muitas versões produzidas.

b) Protótipos descartáveis: São protótipos desenvolvidos de uma especificação inicial. Depois de projetados, são entregues para experimento e descartados após a avaliação. Dessa forma, não podem ser aproveitados para integração ao sistema final. Segundo Wazlawick (2019), os protótipos são gerados apenas para estudar aspectos do sistemas, compreender melhor os requisitos e reduzir riscos, para então serem descartados (BRITTO et al., 2011; WAZLAWICK, 2019).

Como visto nesta seção, a prototipação em geral surgiu como uma ferramenta para o levantamento e a validação dos requisitos de *software* em casos bem específicos, desenvolvido com ferramentas de prototipação, numa linguagem diferente daquela que era usada no produto final e com o protótipo sendo descartado ao final da validação.

No caso aqui tratado, foi através de um *software* de *design* de interface. Assim sendo, seguindo esses conceitos, pode-se classificar os protótipos obtidos (pelo Balsamiq Wireframe) como elemento fundamental para o desenvolvimento deste trabalho como: Protótipos de Baixa Fidelidade e descartável.

2.2 Engenharia de Software com reuso

A Engenharia de Software baseada em reuso é uma estratégia da engenharia de software em que o processo de desenvolvimento de novos *softwares* é orientado para o reuso de *softwares* já existentes. Embora essa abordagem de reuso tenha sido proposta como uma estratégia de desenvolvimento há mais de 40 anos, só nos anos 2000 que o desenvolvimento com reuso se tornou uma norma para novos sistemas de negócios (SOMMERVILLE, 2011).

Segundo Nirajan (2010), reutilização de *software* é o uso de conhecimento da engenharia ou artefatos de componentes de *software* existentes para construir um novo sistema. Existem muitos "produtos" de um trabalho que podem ser reutilizados, por exemplo, código-fonte, projetos, especificações, arquiteturas e documentação. O produto de reutilização mais comum é o código-fonte. Reutilizar trata da capacidade de combinar componentes de *software* independentes para formar uma unidade maior de *software* e para incorporar componentes reutilizáveis em um sistema de *software*. Os programadores devem ser capazes de encontrá-los e entendê-los (NIRANJAN; RAO, 2010).

Há vantagens e desvantagens no reuso de *software*. Uma vantagem é a redução dos custos de desenvolvimento e manutenção dos *softwares*, já que menos componentes de *software* precisam ser especificados, concebidos, implementados e validados. No entanto, a redução de custos é apenas uma das vantagens do reuso. Pode-se citar ainda: maior produtividade no processo de desenvolvimento, aumento da qualidade do *software* e diminuição do tempo/prazo de entrega do *software* (NIRANJAN; RAO, 2010).

Por outro lado, não deixam de existir custos e problemas associados ao reuso, uma vez que existe um custo significativo associado ao processo de compreender se um

componente, ou função, é adequado para o reuso, e em testar "trechos" reutilizados para garantia de sua confiança. Esses custos adicionais significam que as reduções nos custos de desenvolvimento por meio de reuso podem ser menores do que o previsto (SOMMERVILLE, 2011). Além do mais, um dos maiores problemas de reutilização de *software* em muitas organizações é a incapacidade de localizar e recuperar componentes de *software* existentes (NIRANJAN; RAO, 2010).

Porém, verifica-se que as vantagens superam as desvantagens na prática da indústria de *software* porque consegue-se entregar *software* de valor mais rápido para os clientes. Além disso, uma prática comum na indústria de desenvolvimento é o chamado reuso oportunista, também conhecido como *ad hoc* ou copia-e-cola (ASSUNÇÃO; MENDONÇA; VERGILIO, 2018).

O conceito de reuso pode ser incorporado em abordagens como padrões de projeto, produtos configuráveis de sistema e geradores de programa. Quando conceitos são reusados, o processo de reuso inclui atividades nas quais os conceitos abstratos são instanciados para criar componentes reusáveis executáveis, fazendo com que o reuso seja possível em diferentes níveis, desde funções simples até aplicações completas. A Figura 2 abaixo define várias possíveis maneiras de implementação de reuso de *software* (SOMMERVILLE, 2011). E uma delas é que define a abordagem deste trabalho, que é a geração de programas, ou trazendo para o nosso contexto, geração de código.

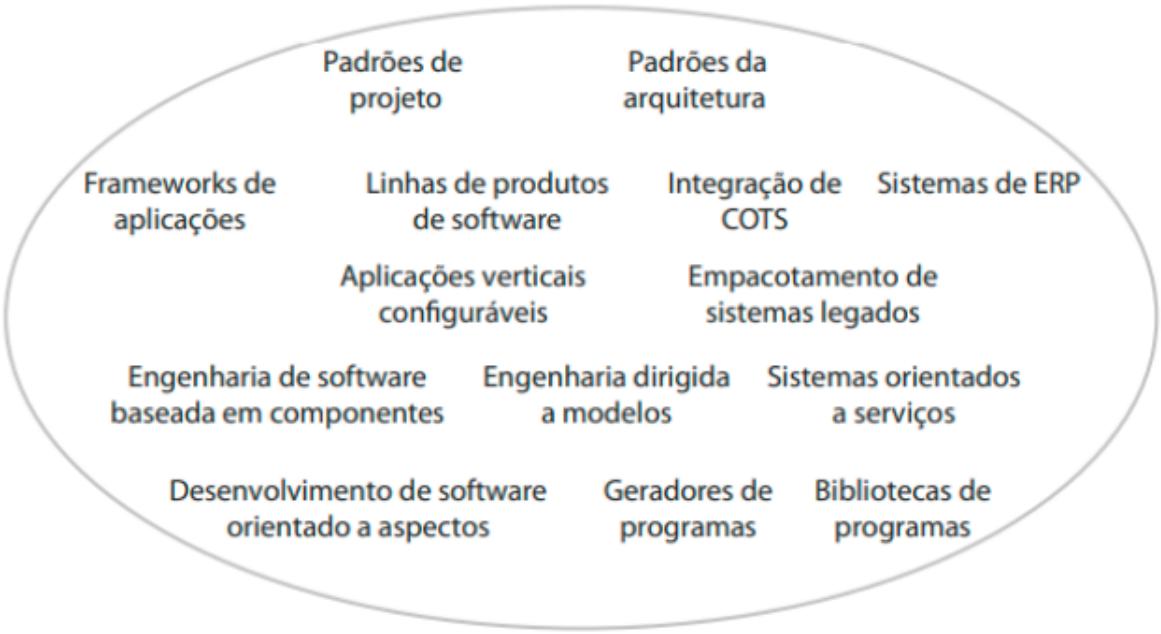


Figura 2 – Os panoramas de reuso.

Fonte: *Sommerville (2011).*

Com isso, neste trabalho a abordagem de reuso adotada foi a de "Geradores de programas", pois como citado nos objetivos, desenvolveu-se uma ferramenta que tem como "produto final" um gerador de código *front-end* para aplicações web, de modo que se tem como *input* arquivos JSON [1] e que foi utilizado dados contidos nesse JSON para ter como saída, código HTML. O reuso entra quando define-se que será utilizado códigos HTML de componentes já definidos e existentes, como *tags* HTML e em especial classes Bootstrap, que será abordado no próximo capítulo.

2.2.1 Geração automática de código

Segundo Moreira (2003), gerador automático de código é uma ferramenta que gera códigos-fontes com o objetivo de facilitar e acelerar os processos de desenvolvimento de sistemas. Mas sabe-se que há geradores para uma grande variedade de aplicações. Contudo, os mais comuns são geradores para interface gráfica de usuário,

[1] JSON (JavaScript Object Notation) é um formato leve de intercâmbio de dados. É fácil para humanos ler e escrever. É fácil para as máquinas analisar e gerar. JSON é um formato de texto completamente independente de linguagem, mas usa convenções familiares aos programadores da família C de linguagens, incluindo C, Python e muitos outros (https://www.json.org/json-pt.html).

como estes que se utilizam como dados de entrada algum outro arquivo ou templates dos códigos a serem gerados. Ele faz parte de um dos tipos do reuso, como mostrado na Figura 2.

De acordo com Monteiro (2016) e Saturno (2007), as vantagens de utilizar uma ferramenta que gera códigos automáticos são:

- Consistência: o código gerado tende a ser mais limpo e claro, reduzindo significativamente a quantidade de erros;
- Customização: caso haja necessidade de mudar a forma de geração, basta alterar o gerador e produzir uma nova versão do código;
- Padronização: os programadores tendem a copiar o estilo do código gerado, criando um padrão para codificação, diminuindo o número de possíveis erros ocorridos devido às diferentes maneiras de implementação;
- Possibilidade de redução de custos do projeto;
- Aumento da produtividade, uma vez que traga garantia da aceleração dos processos de desenvolvimento;
- Abstração da interação com o modelo de dados.

Apesar das vantagens mencionadas, os autores citam ainda desvantagens atribuídas à utilização de geradores de códigos automáticos:

- O código-fonte gerado pode não estar em concordância com o protótipo de codificação do desenvolvedor;
- O código-fonte gerado pode desconsiderar questões de desempenho, otimização, estrutura, integração com outros sistemas ou documentação;
- É viável apenas em um conjunto restrito de aplicações;
- Sempre haverá uma certa quantidade de código que deverá ser escrita manualmente, e esta quantidade varia de aplicação para aplicação;

Além do mais, em Saturno (2007) *apud* (HERRINGTON, 2003) são descritas as etapas necessárias para a construção de um gerador de código. De modo geral, depende da necessidade da aplicação, mas as etapas a seguir são generalizadas:

a) Etapa inicial: Escrever o código de saída manualmente, pois facilita a identificação das informações que são necessárias e que devem ser extraídas da entrada do gerador;

b) Desenvolver o gerador: deve-se, então, determinar a forma de tratamento das informações de entrada, como scripts ou Inteligência Artificial, bem como determinar se as saídas serão geradas através de templates ou embutidas no código do gerador;

c) Desenvolver o processador de entrada: o processador deverá ler os dados de entrada e extrair as informações necessárias para geração da saída;

d) Criar os modelos de saída: o próximo passo é pegar o código desenvolvido na etapa inicial e usá-lo como modelo para a criação do template ou para desenvolver o código que gerará a saída;

e) Desenvolver o processador de saída: a última etapa consiste em pegar as informações extraídas da entrada e gerar o código de saída.

Em Herrington (2003) é afirmado que existem duas classes de geradores de código: Passivos e Ativos. Os geradores passivos geram conjuntos de códigos e não mantêm responsabilidade sobre estes códigos, deixando que os desenvolvedores possam alterar livremente. Um exemplo de gerador passivo são os assistentes de geração das *Integrated Development Enviroments* (IDEs). Por outro lado, geradores ativos de código são responsáveis pelos códigos que geram a longo prazo, ou seja, toda ferramenta onde o código gerado depende de um critério de geração, como um template por exemplo. Isso quer dizer, segundo o autor, que quando mudanças nestes códigos são necessárias, os desenvolvedores devem enviar novos parâmetros para o gerador e também executá-lo novamente para obter o novo código ou a atualização.

O autor cita alguns modelos de geradores de código, seja ativos e passivos, entre os quais são citados:

a) *muging:* gera um ou mais arquivos como saída, a partir de padrões encontrados em arquivos de entrada;

b) *inline-code expander:* gera um ou mais arquivos como saída, a partir de sintaxes "embarcadas" no código dos arquivos de entrada;

c) *mixed-code generator:* é similar ao modelo anterior, porém gera o arquivo de saída sobre o arquivo de entrada;

d) *partial-class generator:* diferente dos modelos citados acima, este não utiliza código fonte e sim uma representação abstrata do código a ser criado, sem a utilização de filtros ou a substituição de fragmentos de códigos;

e) *tier generator:* gera todo o código fonte de uma camada ou seção de uma aplicação. Neste modelo o gerador tem informações suficientes para construir todo o código fonte, incluindo as classes e suas funções (HERRINGTON, 2003).

Com isso, independentemente da classe (passivo ou ativo) ou do modelo, no desenvolvimento de um gerador de código de modo geral deve-se atender as seguintes etapas (HERRINGTON, 2003): identificar a saída desejada, definir a entrada e como a mesma será analisada, interpretar e recuperar as informações da entrada necessárias para gerar a saída e gerar os arquivos de saída a partir da entrada. Analisando essas etapas e as características, a ferramenta de geração de código aqui desenvolvida pode ser classificada como passivo, pois partiu-se do pressuposto que ao gerar um código inicial, ele será adaptado.

2.3 Trabalhos relacionados

Em Monteiro (2016), é apresentado sobre um gerador de código para desenvolvimento de um sistema Web a partir da modelagem entidade-relacionamento – que nada mais é do que uma representação de um problema a ser modelado –, linguagem

de programação e padrões de projeto que o usuário determina. O nome da ferramenta desenvolvida é GCER – Gerador de Códigos para Entidade-Relacionamento e foi desenvolvida em linguagem Java com uso de banco de dados Oracle 11g e tem como destaque a permissão da definição do padrão de projeto a ser utilizado. Os resultados deste foram avaliados através da geração e compilação de códigos de um projeto para cadastro de veículos e a geração com êxito evidencia a viabilidade da ferramenta proposta para a geração automática de códigos no processo de desenvolvimento de *software*.

Em comparação com este trabalho, podemos destacar que ambos têm o objetivo de gerar código para sistema web, ou seja, ter como saída um arquivo HTML. Porém, esse descrito em Monteiro (2016) usa uma metodologia diferente e também ferramentas diferentes, pois é baseado no modelo entidade-relacionamento, uma vez que este aqui é baseado em JSON extraído de protótipos de telas de baixa fidelidade. Outro ponto é a ferramenta para desenvolvimento do gerador, pois em Monteiro (2016) é utilizado JavaServer Face, JQuery e scripts SQL, contando que para funcionamento, é necessário preencher alguns campos na aplicação e salvar uma entidade, essa interação é feita numa interface e tem toda uma questão de pastas e dependência de Banco de dados e outras configurações. Já neste, foi desenvolvido apenas com linguagem Python e uma biblioteca dependente (Pandas) e precisa apenas pôr como *input* no próprio *script* o nome e caminho do arquivo de entrada (JSON) e o de saída que já será possível ter o arquivo HTML gerado nesse caminho definido. De modo geral, este aqui desenvolvido gera código para *front-end*, enquanto o desenvolvido em Monteiro (2016) gera código para persistência de entidades.

Em Saturno (2007), o autor desenvolveu um Plugin com o propósito de realizar a geração de código a partir de um diagrama de telas projetado com o Enterprise Architect (EA). A ferramenta gera código para alguns dos principais componentes usados na construção de interfaces de usuário, como botões, caixas de texto, rótulos e botões de checagem. O código é gerado segundo as diretrizes de um template definido para este fim, o que traz ao usuário uma grande flexibilidade ao processo. De modo geral, a ferramenta gera código fonte para as linguagens Java e Delphi.

Em comparação com este trabalho, as ferramentas utilizadas para desenvolvi-

mento são um tanto diferentes, além de que a saída é com focos diferentes, uma vez que o autor desenvolveu sua ferramenta para gerar código em Java e Delphi e este aqui é para gerar código *front-end* HTML para web. Porém, o que se equivalem é a metodologia utilizada e o processo de desenvolvimento, pois ambos utilizam praticamente a mesma metodologia e também são mapeados componentes usados na construção de interfaces de usuário, como botões, caixas de texto, rótulos e botões de checagem. Diferindo de que o trabalho desse autor foi para classes em Delphi e Java, e este aqui para HTML com Bootstrap.

Já em Asiroglu *et al.* (2019), podemos dizer que é o trabalho que mais se aproxima com o objetivo deste, pois os autores desenvolveram um gerador automático de código HTML a partir de *mock-up* de imagens, desenhados à mão, usando técnicas de aprendizado de máquina. O objetivo é reconhecer os componentes criados no desenho do *mock-up* e codificá-los de acordo com a hierarquia da página da web. Esses *mock-ups* foram obtidos do dataset público do Github da Microsoft AI Labs. O método de funcionamento é basicamente este: a detecção de objeto é aplicada na imagem de entrada com técnicas de processamento de imagem. Após esta etapa, os objetos identificados são colhidos e, em seguida, os componentes obtidos são marcados com o modelo treinado da CNN. Por fim, a saída deste modelo é convertida em código HTML por meio do HTML Builder, que é onde os componentes reconhecidos são traduzidos para Código HTML por meio de estrutura de bootstrap.

Em comparação com este trabalho, algumas coisas são comuns como a finalidade de gerar código front-end HTML com Bootstrap, tomar como base mock-ups de telas e o mapeamento de componentes HTML. O que difere é o fato de que em Asiroglu *et al.* (2019) a abordagem para resolução do problema foi utilizar Inteligência Artificial através do reconhecimento de imagens de *mock-ups* feitos à mão. Já neste aqui, como um arquivo JSON é utilizado como input, pode-se prever que já tenha mais informações sobre os elementos da tela e com isso faz com que não haja necessidade fazer reconhecimento de imagem através de Inteligência Artificial. Além do mais, a ferramenta aqui desenvolvida pode ser utilizada em times de desenvolvimento sem a necessidade de gerar ainda uma base de dados ou treinar o modelo com *mock-up*, tornando-se assim uma solução mais

simples de aplicação direta na indústria.

2.4 Tecnologias utilizadas

As aplicações web ficam cada dia mais sofisticadas e complexas, pois com o aumento do uso da internet, há uma certa concorrência de conteúdos e é preciso prender a atenção dos usuários em uma página e envolvê-los com experiências interativas. O desenvolvimento desse tipo de aplicação envolve também o desenvolvimento de elementos de interface caracterizado como *front-end* e de ferramentas e recursos que dão suporte e facilitam o processo de desenvolvimento tanto *front-end* quanto *back-end*.

O ***Frond-End*** pode ser definido como a parte visual de um site, é a primeira camada na qual nos deparamos em um *software* com a finalidade de nos transmitir informações visuais, ou seja, aquilo que conseguimos interagir. Quem trabalha com *front-end* é responsável por desenvolver por meio de código uma interface gráfica, normalmente com as tecnologias base da Web como HTML, CSS e JavaScript. Já o ***Back-End***, como o próprio nome sugere, vem da ideia do que tem por trás de uma aplicação, pois o *back-end* trabalha em boa parte dos casos fazendo a ponte entre os dados que vem do navegador rumo ao banco de dados e vice-versa, sempre aplicando as devidas regras de negócio, validações e garantias em um ambiente onde o usuário final não tenha acesso e possa manipular algo (SOUTO, 2019).

Na academia, pode-se dizer que a prototipação é a melhor forma de propor uma solução adequada para o problema de um cliente na vida real. De modo geral, o usuário poderá avaliar como os recursos estarão distribuídos, a organização do *layout* e outros itens que impactam na experiência de uso. A partir do protótipo (principalmente o *front-end*), ajustes poderão ser feitos no projeto para adequar às expectativas e alinhar aos objetivos. Dessa forma, a qualidade final da ferramenta é otimizada, pois se temos um bom projeto "prototipado", as chances de desenvolver o real projeto e ter vantagens em relação a tempo e custo são altas.

Com isso, o trabalho aqui proposto se utilizou de algumas ferramentas, ou tecnologias, em diferentes etapas, como o programa de prototipação Balsamiq Wireframes,

o editor de código (ou IDE) Visual Studio Code, a linguagem de programação Python e a biblioteca Pandas, e para a parte web o HTML, CSS e o framework Bootstrap. A seguir, está listado a definição de como funciona cada tecnologia usada, para, no próximo capítulo, entendermos como elas foram utilizadas para o desenvolvimento deste trabalho.

2.4.1 HTML, CSS e Bootstrap

O HTML, do inglês *Hyper Text Markup Language*, é o bloco de construção mais básico da web. Isso quer dizer que ele permite a construção de *websites* e a inserção de novos conteúdos, como imagens e vídeos, por meio dos hipertextos (MDN contributors, 2005).

O termo *Hyper Text*, "Hipertexto"no português, se refere aos *links* que conectam páginas da Web entre si, seja dentro de um único site ou entre sites. Ao carregar conteúdo na Internet e vinculá-lo a páginas criadas por outras pessoas, nos tornamos um participante ativo na *world wide web* (www). Um elemento HTML é separado de outro texto em um documento por "tags", que são códigos que orientam a estrutura do documento, como o tamanho, a fonte e as quebras de linhas e consistem no nome do elemento entre ¨< >". O nome de um elemento dentro de uma tag pode ser escrito em maiúsculas, minúsculas ou uma mistura. Por exemplo, a *tag* <title> pode ser escrita como <Title>, <TITLE> ou de qualquer outra forma (MDN contributors, 2005).

Apesar de o HTML ser uma linguagem que possui diversas funções, ela não consegue criar estruturas dinâmicas de conteúdo, por isso, em geral, os *websites* costumam incluir duas outras linguagens: CSS e JavaScript. Enquanto o primeiro fica responsável por incluir cor, *layout* e animações, o JavaScript permite a inclusão de galerias de fotos e *pop-ups*, por exemplo (MDN contributors, 2005).

O CSS (*Cascading Style Sheets* ou Folhas de Estilo em Cascata) é uma linguagem de estilo usada para descrever a apresentação de um documento escrito em HTML ou em XML e descreve como elementos são mostrados na tela, no papel, na fala ou em outras mídias (MDN contributors, 2005).

Seguindo na mesma linha de tecnologias, foi utilizado também o Bootstrap, que é um *framework* – ferramenta – de *front-end* de código aberto para desenvolvimento HTML, CSS e JS e que serve para personalizar e projetar sites responsivos. Ele contém todos os tipos de *templates* baseados em HTML e CSS para várias funções e componentes, como por exemplo, navegação, sistema de grades, carrosséis de imagens e botões (OTTO; THORNTON, 2011).

Ainda que os Bootstraps realmente poupem o tempo que os desenvolvedores gastam para gerenciar e criar os *templates*, o objetivo principal dele é criar sites responsivos, mas para nosso contexto, utilizamos para evitar de utilizar muito CSS e pelo fato de já estar atrelado muitas vezes nas próprias *tags* HTML, o que facilita e a escrita do código e foi a solução mais prática para o desenvolvimento da ferramenta em questão, onde se entenderá melhor no próximo capítulo.

2.4.2 Balsamiq Wireframes

O Balsamiq Wireframes é um *software* que permite construir interfaces gráficas simplificadas, como *mock-ups* [2] e protótipos de sites para web e, hoje em dia, também aplicativos móveis. Ele funciona de modo que permite ao "designer" (utilizador) arranjar blocos pré-construídos de *widgets* e elementos. Em resumo, é uma ferramenta rápida de *wireframes* de IU de baixa fidelidade que reproduz a experiência de esboçar ou desenhar em um bloco de notas ou quadro branco, mas usando um computador (Balsamiq Wireframes, 2008). Além disso, é possível exportar o protótipo gerado em diversos formatos, como em PNG e JSON.

O programa está disponível diretamente no site do Balsamiq [3] e pode ser encontrado para download em uma versão limitada para Desktop e também é possível utilizá-lo na nuvem. No entanto, é possível adquirir uma licença da versão completa para estudantes e professores, esta que solicitamos para utilizarmos neste trabalho.

[2] Mock-up: Consiste em uma representação de um determinado projeto, que pode ser feita em escala ou em tamanho real. Ele é aplicado na apresentação de ideias de maneira mais elaborada (https://www.futuraexpress.com.br/blog/o-que-e-mockup/).

[3] Site do Balsamiq: https://balsamiq.com/wireframes/

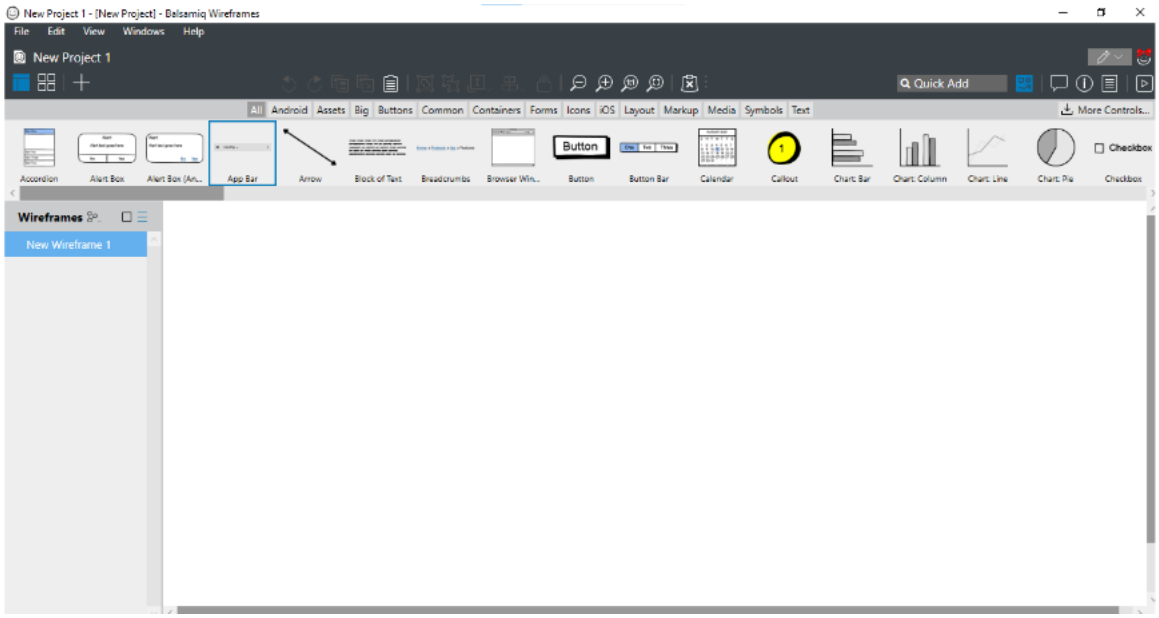


Figura 3 – Interface Inicial do Balsamiq Wireframes (versão Desktop)

2.4.3 Ferramenta Pandas – Python

A ferramenta desenvolvida neste trabalho consiste num script que foi desenvolvido na linguagem de programação Python e com a biblioteca Pandas, que é uma ferramenta – biblioteca – de análise e manipulação de dados de código aberto rápida, flexível e fácil de usar, construído com base na própria linguagem de programação Python (Pandas, 2009). Isso porque a ferramenta permite ler, manipular e agregar dados facilmente, uma vez que temos um arquivo JSON como entrada.

Para utilizar o Pandas, é necessário primeiramente já ter o Python instalado no computador, com isso, através de uma simples linha de comando, o Pandas é instalado, podendo assim, no editor de código, ser importado e utilizado. Existem muitas funções nativas para ler seus dados, normalmente iniciando por pd.read_[extensão], então, neste caso foi utilizado:

```
1  import pandas as pd
2
3  dado = pd.read_json('nome_do_arquivo.json')
```

Figura 4 – Declaração do pandas e uso no editor de código

2.4.4 Visual Studio Code (VS Code)

Para o desenvolvimento do script, para visualização dos arquivos .json e .html e uso de modo geral, foi utilizado o programa Visual Studio Code (VSCode) como IDE, que é um editor de código-fonte bastante popular e que foi desenvolvido pela Microsoft. Ele vem com suporte integrado para várias linguagens de programação e tem um rico ecossistema de extensões (Microsoft, 2015). Ele também é gratuito e está disponível para download no próprio site [4] e tem suporte para vários sistemas operacionais e em várias versões. Vejamos a seguir na Figura 4 a tela inicial de um projeto no VS Code.

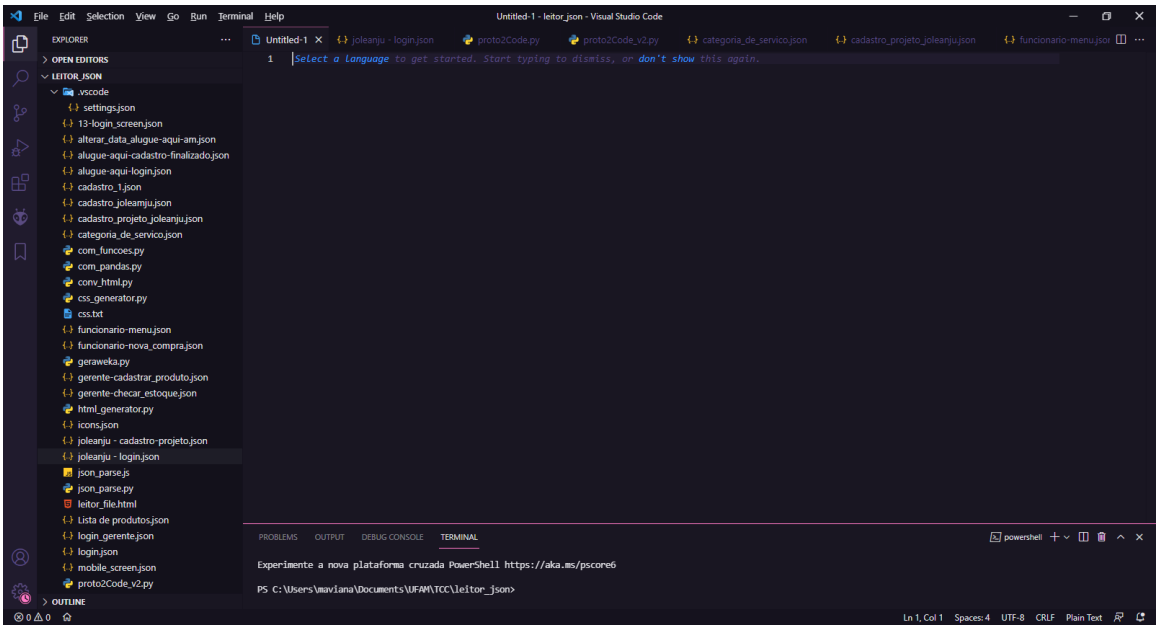


Figura 5 – Tela inicial do VS Code

[4] Site: https://code.visualstudio.com/

3

PROTO2CODE

Este capítulo apresenta a proposta da ferramenta, na qual foi nomeada como Proto2Code, que tem como objetivo acelerar e facilitar o desenvolvimento de aplicações web a partir de análises de protótipos de interface realizados em ferramentas de prototipação de mock-ups.

Esta ferramenta é um script desenvolvido em Python e tem como entrada o arquivo JSON exportado do *software* de prototipação Balsamiq Wireframes e tem como saída o arquivo HTML correspondente a uma interface de tela. Nas seções a seguir está descrito sobre a metodologia de desenvolvimento da ferramenta e os artefatos utilizados.

3.1 Etapas de desenvolvimento

Esta seção apresenta em detalhes a Etapa 3 da metodologia descrita na seção 1.3 do Capítulo 1. A etapa de desenvolvimento da ferramenta é baseada na metodologia descrita em Saturno (2007) *apud* (HERRINGTON, 2003) e foi ajustada ao contexto da ferramenta desenvolvida neste trabalho.

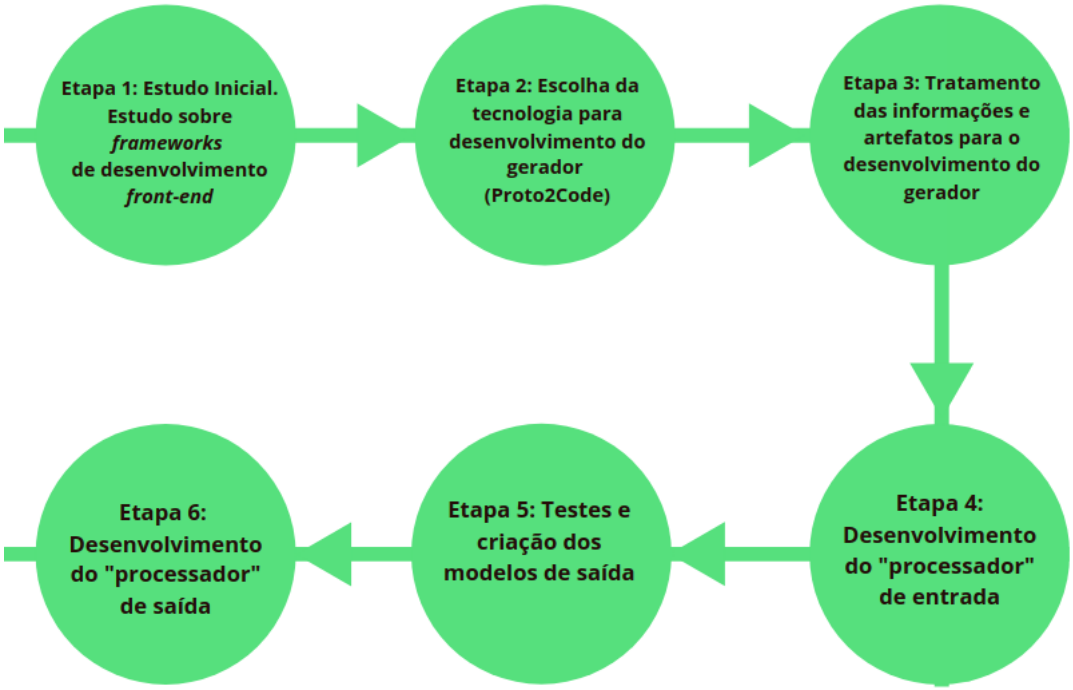


Figura 6 – Etapas da metodologia de desenvolvimento da ferramenta

3.1.1 **Etapa 1:** Inicial

Foi feito um estudo a respeito de frameworks que atualmente desenvolvedores *front-end* utilizam como auxílio para desenvolver sistemas web, tais como Vue JS e React. Além de entender como trabalhar com Node JS [1], JavaScript em conjunto com HTML e CSS para desenvolvimento de *front-ends*.

Para esse aprendizado, escreve-se o código de saída manualmente referente aos protótipos de telas já obtidos no Balsamiq. Esta atividade foi útil para facilitar a identificação das informações que são necessárias e que devem ser extraídas da entrada do gerador. Como foi decidido que o tipo de arquivo a ser tomado como input para o gerador seria o JSON exportado dos projetos do Balsamiq, então começou-se a pensar em qual linguagem de programação o *script* do gerador seria feito.

[1] Node JS: É um *software* de código aberto, multiplataforma, baseado no interpretador V8 do Google e que permite a execução de códigos JavaScript fora de um navegador web. (https://nodejs.org/en/)

3.1.2 **Etapa 2:** Escolher a tecnologia para desenvolver o gerador

A ideia inicial foi desenvolver o gerador com Javascript, utilizando Node JS, ambos já estudados antes. Porém, deparou-se com a dificuldade de ler e manipular estruturas complexas de JSON, como o gerado pelo Balsamiq, que tem uma estrutura de: uma lista de objetos dentro de um objeto que está dentro de um objeto maior, conforme mostra a Figura 7. Então acreditou-se ser mais viável e fácil percorrer essa estrutura toda até chegar nos objetos dentro da lista utilizando Python com a biblioteca Pandas. Desse modo, foi descartado o uso do gerador com JavaScript.

Figura 7 – Estrutura padrão do JSON gerado pelo Balsamiq

Assim como é visto na seção 2.4.3 do Capítulo 2, o Pandas é uma biblioteca de análise e manipulação de dados de código aberto rápida, flexível e fácil de usar, construído com base na própria linguagem de programação Python. Isso porque a ferramenta permite ler, manipular e agregar dados facilmente, uma vez que temos um arquivo JSON como entrada. O nome Pandas é derivado de panel data (dados em painel) (MULINARI, 2021).

3.1.3 **Etapa 3:** Desenvolvimento do gerador: Artefatos e mapeamentos

Nesta etapa, começou-se então a determinar a forma de tratamento das informações de entrada, bem como determinar se as saídas seriam geradas através de templates ou embutidas no código do gerador.

Dessa maneira, foi feito um levantamento e análises dos componentes disponíveis no Balsamiq, pois todos eles têm uma estrutura fixa que é passada/reportada no JSON, como se fosse sua a identidade, contendo um nome, um ID e seu tipo. Esse mapeamento foi feito numa planilha em excel de modo que foi separado em 4 colunas: uma coluna para todos os tipos de elementos disponíveis, o que se denomina 'typeID'.

Na segunda coluna era o código HTML pesquisado e equivalente para aquele tipo de elemento. Na terceira coluna era o tipo da plataforma, pois tem tanto elementos para web quanto mobile, e o foco do mapeamento era os de web, então os para mobile foi deixado identificado para não ser mapeado. A última coluna se refere ao símbolo/componente disposto no Balsamiq para facilitar na hora das pesquisas e fazer a associação visual do typeID com seu componente.

A Figura 8 abaixo mostra uma parte da organização da planilha e toda a planilha pode ser vista no Apêndice A:

TypeID - JSON	HTML	Componente Balsamiq
SubTitle		A Subtitle
TextInput	<input type="text">	ou
Button	<button type="button" class="btn btn-primary">Primary</button>	Button
Label	<label> </label>	Some text
Title		A Big Title
Accordion	<button class=" ">Section 1</button> Text 1 <button class=" ">Section 2</button> Text 2 ...	Item One Item Two Item Three
CheckBox	<input type="checkbox" id="Check" onclick="Function()">	Checkbox
Etc...	Etc...	Etc...

Figura 8 – Exemplo da planilha de mapeamento dos componentes do Balsamiq

3.1.4 **Etapa 4:** Desenvolvimento do “processador” de entrada

Denomina-se como “processador” de entrada a parte do código onde deve ser capaz de ler os dados de entrada, neste caso o JSON, e extrair as informações necessárias para manipulação e tratamento dos dados, através de uma lógica, para então chegar na próxima etapa, que resumidamente é gerar o arquivo HTML.

Desse modo, neste primeiro momento o objetivo foi desenvolver um script inicial que lesse o JSON e imprimisse os elementos dele de forma simples, em formato de string, antes de partir para a etapa do tratamento desse JSON. O script percorre a estrutura do JSON e imprime seus elementos principais e também as suas propriedades.

A partir desse momento, começou-se a pensar na lógica para ler cada ‘typeID’

do JSON e associá-lo a um código ou tag HTML. A questão era: Como armazenar esses códigos ou tags associado a um certo componente pelo seu 'typeID' identificado? Com isso, foi pensado no dicionário, que em python é um tipo de estrutura de dados do tipo coleção, ou seja, um objeto que contém mais que um valor (SPAK, 2019).

Diferentemente das listas em que os elementos são acessados através de uma posição ou índice, nos dicionários o acesso às informações ocorre através de chaves. Os elementos de um dicionário são armazenados de forma não ordenada e utilizam uma estrutura de pares contendo chave e valor ou também encontrado na literatura como {key : value}, de modo que seus elementos sejam exclusivos, embora os elementos de um valor possam ser duplicados, uma vez que a chave é diferente (SPAK, 2019). Trazendo para o nosso contexto, teríamos então um dicionário com estrutura {'typeID': 'código HTML'}.

3.1.5 **Etapa 5:** Criação dos modelos de saída

O próximo passo agora é pegar o código desenvolvido na etapa anterior e usá-lo como modelo para a criação do template ou para desenvolver o código que gerará a saída.

O *script* desenvolvido na etapa anterior foi a versão inicial e a implementação básica para começar a testar o JSON de alguns protótipos já prontos. Com isso, à medida que se ia testando, ia sendo identificado alguns pontos faltantes *(gaps)* e consequentemente melhorias iam sendo feitas no *script*. A ideia era também tentar extrair as informações sobre a posição dos componentes na tela e adaptar para o CSS, tanto que foi feito um levantamento em uma planilha também correlacionando o typeID do elemento, o seu HTML mapeado e o CSS "compatível", porém não foi possível ao identificar alguns problemas como os destacados abaixo:

- Não saber a unidade de medida dos parâmetros dispostos no JSON, pois para CSS é tido como pixel – px, como o width e height.
- Não podia ser definida uma classe no HTML de um elemento que satisfizesse o CSS, pois para uma outra tela, poderia não satisfazer, então de modo geral

um grande desafio ainda é manter classes genéricas para não se limitar a um protótipo.

- Pode-se dizer que o maior problema foi sobre as cores, pois no JSON é gerado um código numérico interno representando cores, links e imagens, já no CSS é código hexadecimal e para imagens é através de links.

Desse modo a abordagem foi utilizar o *framework* bootstrap, como já definido na seção 2.4.1 do Capítulo 2. Isso porque o *framework* permite ter *templates* de classes já estilizadas, através de *tags* na própria estrutura HTML, como por exemplo, um botão, que pela classe button já se tem um elemento encapsulado, isso quer dizer que já vem com um tamanho definido e uma cor definida, podendo, claro, ser alterado pelo desenvolvedor posteriormente.

3.1.6 **Etapa 6:** Desenvolvimento do "processador" de saída

Nesta etapa, por fim, consiste em pegar as informações extraídas da entrada (etapas 4 e 5) e gerar o código de saída, isso através do *script* que é denominado "processador de saída".

Uma vez que o dicionário foi populado, o acesso aos elementos pode ser realizado de modo similar ao método de acesso das listas. Entretanto, aqui é passado a chave para qual se deseja acessar o valor, e isso foi feito na lógica do *script* em si, pois tinha-se o typeID e os códigos HTML, então o desafio agora era como fazer esse tratamento após ler o JSON e ter identificado seu typeID no dicionário.

A solução foi usar estruturas condicionais *(if | else)* e de repetição *(for)* junto com a função *get()* para 'capturar' dados do JSON e tratar no restante do *script*. Como não se tem apenas elementos de mesmo nível nos objetos de cada componente no JSON, então foi preciso pensar sempre na forma de como acessar uma camada mais abaixo: o de propriedades (properties).

Assim, resumidamente, a lógica desse *script* é: passar pela estrutura de objetos e da lista e "pegar" o que está em typeID e armazenar em type_id_match toda em minúscula. Assim, se dentro do objeto ele encontrar os elementos, então deve substituir

o conteúdo do seu elemento para dentro da string criada, onde pode-se ver entre chaves, por exemplo '{SRC}', senão, é acessado as propriedades e substituído o texto que estiver no JSON.

Esses dados que são trazidos do JSON entre chaves, são os dados genuínos que estarão no código HTML, por isso no trecho de código do dicionário, vê-se muitos elementos com {TEXT} dentro de uma tag, pois lá estará a mesma informação contida no JSON quando se gerar o arquivo .html. E é isso que faz parte do processo de facilitação para o possível desenvolvedor, pois o mesmo não precisará escrever textos e informações básicas da tela porque o arquivo .html gerado pelo Proto2Code já lhe trará isso.

3.2 Uso da ferramenta

Esta seção trata de como utilizar o Proto2Code para gerar um código HTML referente a uma interface de protótipos do Balsamiq. Ela basicamente pode ser explicada em 6 etapas simples. A Figura 8 mostra o processo de uso e as seguintes etapas na sequência:

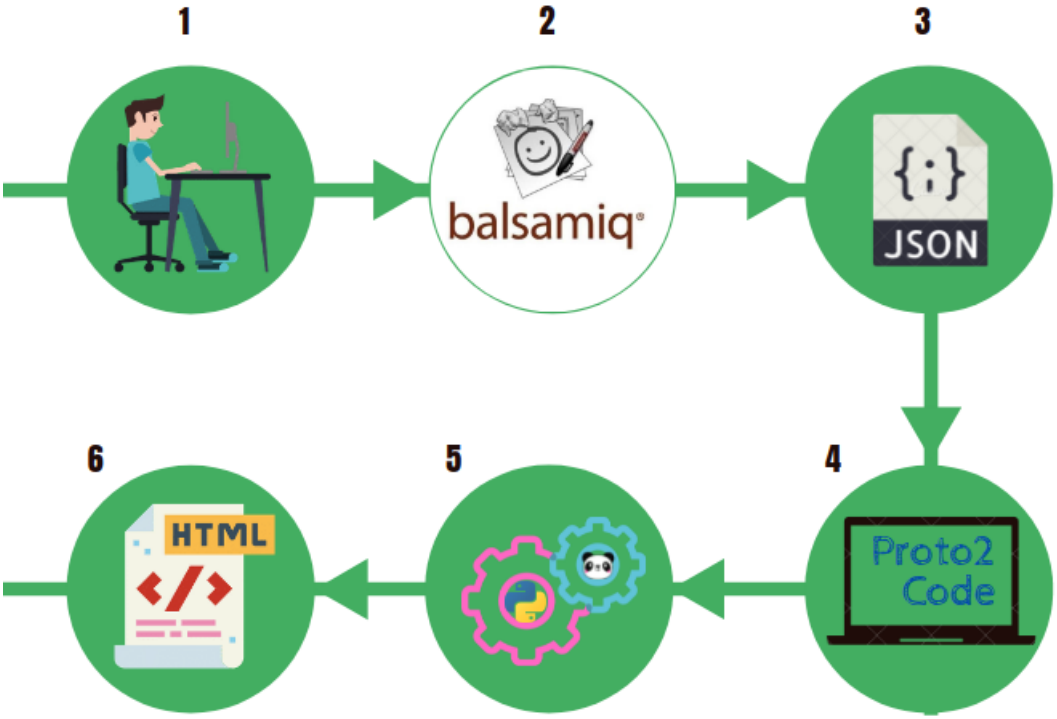


Figura 9 – Processo de uso do Proto2Code

- Em **1**, refere-se ao desenvolvedor (ou usuário em geral) que utilizará o Proto2Code. Ele pode ser dois atores diferentes, pois pode se tratar do mesmo que desenvolve o(s) protótipo(s) no Balsamiq para depois, enfim, dar seguimento ao processo de uso ou pode apenas ter em mãos os protótipos já feitos por terceiros no Balsamiq.

- Em **2**, refere-se ao projeto do protótipo no Balsamiq Wireframes, pois deve-se ter projetos das interfaces de *front-end* que se deseja gerar em HTML. Uma observação importante sobre esses projetos com relação ao desenvolvimento do Proto2Code é que os protótipos devem estar com os elementos postos no Balsamiq em ordem. Caso contrário, o Proto2Code obedece a ordem que os elementos foram colocados na tela e gera desordenado também. Uma outra observação é que deve-se evitar usar elementos que são classificados como Mobile no Balsamiq, pois muitos não foram mapeados para o desenvolvimento do Proto2Code.

- Em **3**, é o passo em que o usuário já está com o projeto do Balsamiq aberto, conforme descrito no passo 2, e agora deve exportar o arquivo JSON de cada tela. Uma observação é que ao exportar (File > Export > Wireframe to JSON), o programa não sugere como e onde exportar, apenas mostra um pop-up de que os dados requeridos estão prontos para serem colados onde o usuário quiser, seja num arquivo, em um email etc. Então o usuário ainda tem o trabalho, manualmente, de colar a estrutura exportada em um editor de texto de preferência e salvar como arquivo JSON.

- Em **4**, trata-se de abrir o arquivo em Python, na IDE de preferência do usuário, onde se está todo o script que define o Proto2Code e ajustar os *inputs*/caminho do arquivo JSON e o nome desse arquivo e também pôr o nome do arquivo que se deseja gerar em HTML e seu caminho.

- Após 4, o **5** basicamente é quando o usuário clica no compilador Python (RUN) e o script é executado em alguns milisegundos.

- E por fim, em **6**, o usuário deve ir até o caminho que foi definido para salvar o arquivo e ter acesso ao arquivo .html gerado pelo Proto2Code.

4

AVALIAÇÃO

Este capítulo apresenta a avaliação realizada com a ferramenta Proto2Code. Esta avaliação tem como objetivo verificar se a ferramenta auxilia/acelera o processo de criação de aplicações web na prática, quando se trata de *front-end*. Considerando que foram feitos testes no ato do desenvolvimento da ferramenta, pois à medida que se ia desenvolvendo o *script* (processador de entrada e processador de saída), o mesmo já era testado e usado para gerar alguns códigos para analisar como estava a saída com determinado *input*. Essa avaliação foi conduzida como um estudo piloto para validar a ferramenta e envolveu dois voluntários para verificar o tempo e o esforço na elaboração da interface usando a ferramenta desenvolvida, Proto2Code.

De modo geral, essa avaliação busca responder a seguinte questão: É viável a utilização do Proto2Code como auxílio num ambiente de desenvolvimento de *software*? O detalhamento desses estudos está descrito nas seções a seguir.

4.1 Estudo piloto

Neste estudo o objetivo foi validar o uso e o tempo de desenvolvimento de modo que foi feito a comparação desse tempo de desenvolvimento entre uma tela gerada sem ferramenta alguma e a mesma tela desenvolvida com uso do Proto2Code. Este primeiro estudo, chamado de "Estudo Piloto", é melhor representado pelas figuras abaixo e é descrito nas próximas seções.

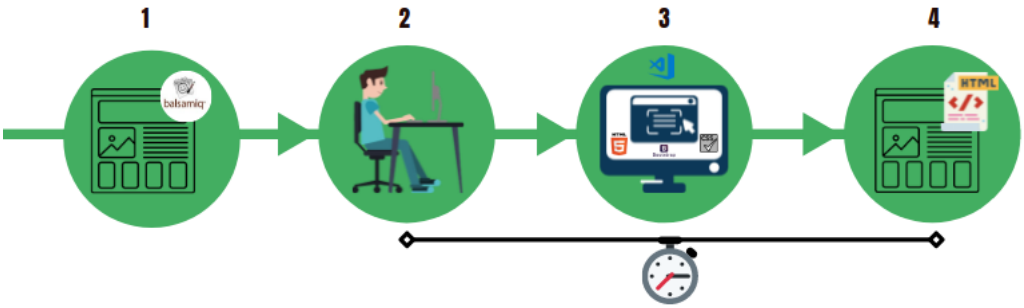


Figura 10 – Processo etapa 1 do estudo piloto

A figura acima se refere à etapa 1 do estudo piloto, onde os números de 1 a 4 identificam a sequência realizada. A etapa é melhor explicada na seção 4.1.2.

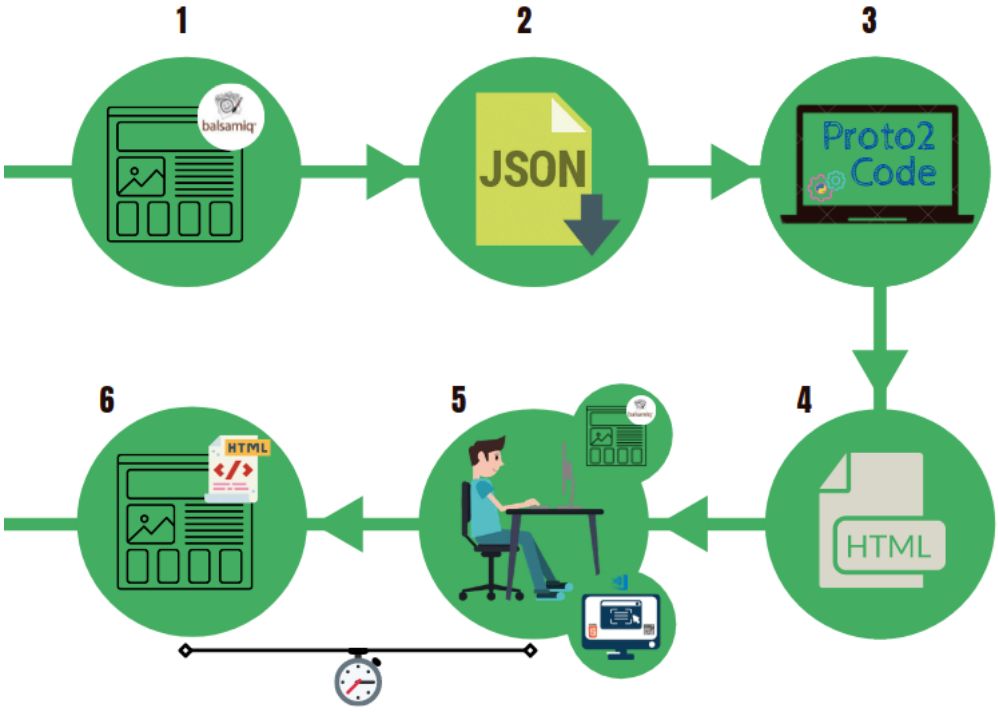


Figura 11 – Processo etapa 2 do estudo piloto

Já a Figura 11, se refere à etapa 2 do estudo piloto, a qual já se tinha incluído o uso do Proto2Code. Os números de 1 a 6 também identificam a sequência realizada nesta etapa e é melhor explicada na seção 4.1.2 adiante.

4.1.1 Preparação

Participaram do estudo piloto dois voluntários da área de desenvolvimento *front-end*. Um dos voluntários é desenvolvedor *front-end* e o outro é desenvolvedor *full-stack* (que atua tanto em projetos *front-end* quanto em projetos de *back-end*). O primeiro voluntário, no qual será citado como "Voluntário 1", é formado em Análise e Desenvolvimento de Sistemas e trabalha na área de *Front-End* há 3 anos. O segundo voluntário, "Voluntário 2", ainda não possui formação acadêmica, mas já trabalha na área de desenvolvimento de software há 5 anos e atualmente é desenvolvedor *Full-Stack*. Após contatados, ambos concordaram em participar do estudo e após isso foi elaborado um TLCE – Termo de Consentimento Livre e Esclarecido. Enquanto isso, foi explicado a eles do que se tratava o estudo e o que precisava ser feito.

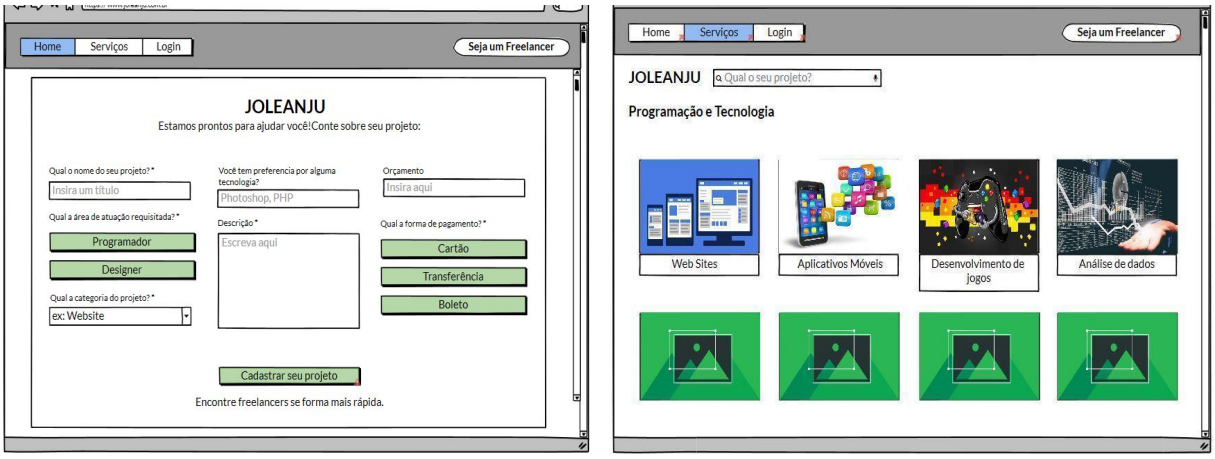
Após concordarem em ajudar, alguns protótipos de baixa complexidade desenvolvidos no Balsamiq por alunos de graduação de uma disciplina do curso de Engenharia de Software, foram filtrados e alguns designs de telas foram selecionados. Esses designs de telas foram salvos como imagens e então enviadas para que os voluntários pudessem selecionar dentre essas telas, a que gostariam de desenvolver.

Os nomes dos protótipos foram anonimizados para fim deste trabalho. Então serão aqui denominados de "Protótipo A", "Protótipo B" etc. Os protótipos de telas selecionados se referem a:

I "Protótipo A" – Este se trata do projeto de um site para contratar programadores e designers para projetos de desenvolvimentos de *softwares*. As telas selecionadas foram as seguintes:

A tela apresentada na Figura 12(a) se trata de uma tela de cadastro, relativamente simples apenas com botões e campos de *input* de texto.

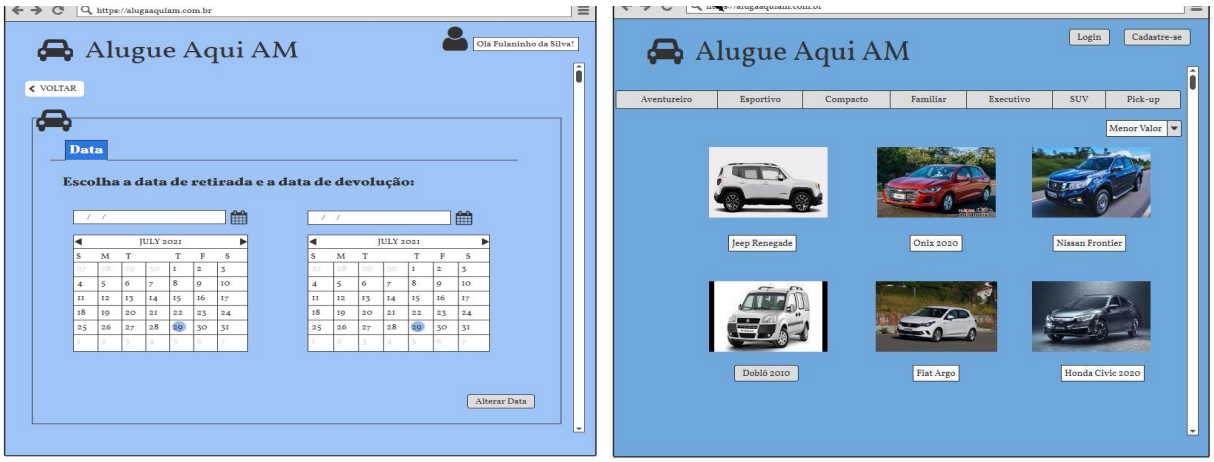
Já a tela apresentada em 12(b) se trata da tela inicial desse site, onde é possível ver alguns *cards* de categorias para o possível usuário escolher uma tecnologia.



(a) (b)

Figura 12 – Telas do Protótipo A

II "Protótipo B" – Este se trata de um site de aluguel de carro. Este protótipo tem uns componentes diferentes, por isso foi selecionado para testes e para o estudo inicial.



(a) (b)

Figura 13 – Telas do Protótipo B

A tela em 13(a) se trata de uma tela de marcação de data de retirada e de devolução do carro alugado. Nela podemos observar basicamente dois campos de data com um calendário embaixo.

Já a tela em 13(b) se trata da tela inicial desse site de aluguel de carros, tal que é composto por uma barra de menu, os botões para login ou cadastro e uns carros em catálogo.

4.1.2 Execução

Antes dos dois voluntários começarem a desenvolver as telas, foi explicado a eles sobre o TCLE e essa concordância foi registrada através de suas assinaturas.

Nota: Esse processo do estudo é visto na Figura 11, onde os números de 1 a 6 da figura serão identificados nos passos entre parênteses, por exemplo: passo 1 da Figura 11 será atribuído como (1) no texto para não interromper a sequência da explicação.

A primeira etapa, representada pela Figura 10, se tratou do desenvolvimento da tela a partir do início, de modo que, baseado no seu protótipo de tela escolhido (1), fosse criado por eles um arquivo .html (2), onde estaria o código conforme suas práticas de desenvolvimentos (3) para enfim ter a tela final (4). Observa-se que entre (2) e (4) há um relógio. Isso simboliza que a partir desse desenvolvimento o tempo estava sendo conferido.

Quanto ao processo da Figura 11, assim que os voluntários escolheram o protótipo (1), foi gerado o respectivo JSON dessa determinada tela pelo protótipo salvo – no Balsamiq (2). Então na sequência foi usado o Proto2Code (3), uma vez que se tem como entrada o JSON, para gerar o arquivo .html correspondente à tela em desenvolvimento (4). Tendo isso, esse arquivo .html foi enviado para os voluntários.

O primeiro voluntário escolheu a tela mostrada na Figura 12(b), enquanto o segundo voluntário escolheu a tela mostrada na Figura 13(a).

Foi pedido para que eles cronometrassem o tempo, tanto do momento em que começaram a criar a tela sem o auxílio da ferramenta até o término quanto do momento em que pegaram o arquivo .html gerado pelo Proto2Code e desenvolveram através dos ajustes a adaptação do código.

Já a segunda etapa se tratou de pegar o arquivo .html já gerado pelo *script* do Proto2Code e aproveitar a sua estrutura básica, como também reorganizar algumas *tags*, identar boa parte do código, reposicionar linhas de códigos, criar suas funções em css etc. Esses ajustes foram necessários para ter a estrutura dos elementos bem posicionados conforme o desejado.

Então, no seu ambiente de desenvolvimento (editor de código e/ou IDE), os voluntários desenvolveram as telas usando HTML, bootstrap e CSS para estilização (5).

Na seção 4.1.3 é visto as telas desenvolvidas por ambos, que representa a ultima etapa (6) do proceso da Figura 11. Além disso, são mostrados os tempos medidos para os dois casos.

4.1.3 Resultados

Aqui será mostrado e discutido sobre os tempos de desenvolvimento das telas de cada etapa do processo do estudo piloto.

Na primeira etapa do estudo piloto (Figura 10), o Voluntário 1, na qual a tela é apresentada na Figura 14, afirmou que desenvolveu a tela **sem** o auxílio do Proto2Code em 40 minutos. Enquanto o Voluntário 2 disse ter desenvolvido a tela da Figura 15 **sem** o Proto2Code em 48 minutos.

A tela desenvolvida referente ao primeiro protótipo ficou da seguinte maneira:

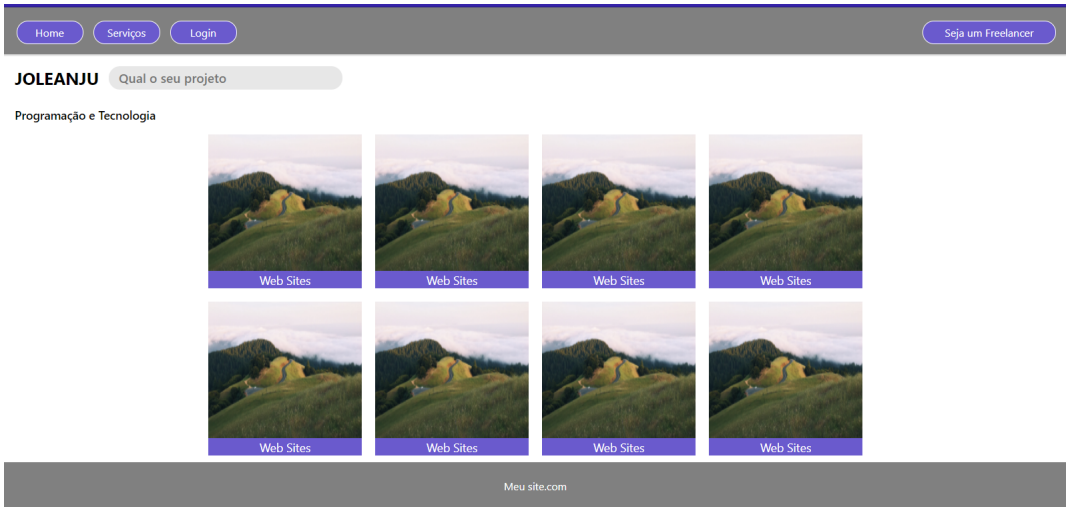


Figura 14 – Tela desenvolvida desde o início pelo Voluntário 1

Enquanto à tela desenvolvida referente ao segundo protótipo, da Figura 13(b), ficou da seguinte maneira:

Figura 15 – Tela desenvolvida desde o início pelo Voluntário 2

Na segunda etapa do estudo piloto (Figura 11), quando os voluntários tiveram que desenvolver a mesma tela, porém agora a partir do arquivo .html que foi enviado para eles gerado pelo *script* que compõe o Proto2Code, obtivemos os seguintes tempos: O voluntário 1 desenvolveu a tela em 1 hora e 10 minutos, enquanto o voluntário 2 disse ter desenvolvido a tela em 35 minutos.

Para exemplificar melhor esses resultados, podemos analisar a Tabela 1 com os tempos cronometrados por cada voluntário.

.	Tempo de desenv. sem o auxílio do Proto2Code	Tempo de desenv. com auxilío do Proto2Code
Voluntário 1	40 minutos	1 hora 10 minutos
Voluntário 2	48 minutos	35 minutos

Tabela 1 – Comparação dos tempos de desenvolvimento de telas

Desse modo, vemos que há uma divergência de tempo. Uma vez que o Voluntário 1 um levou mais tempo para desenvolver a mesma tela com o auxílio do Proto2Code. Já por outro lado, o Voluntário 2 levou um pouco menos de tempo com o auxílio do Proto2Code. Isso deixa algumas opiniões e conclusões que serão discutidas e expostas na próxima seção.

4.2 Discussões

A partir dos resultados apresentados no estudo piloto, tem-se como base a Tabela 1 sobre a comparação dos tempos de desenvolvimento sem o auxílio do Proto2Code e com o auxílio do Proto2Code para esta discussão que trará concepções para as conclusões deste trabalho.

O Voluntário 1 disse que o arquivo enviado não estava com algumas *tags* HTML básicas corretas, o que levou a ter que refazer boa parte da estrutura do código. Isso acontece porque o código gerado pela ferramenta é estático, ou seja, é difinido manualmente e está "fixo"para cada componente. A ordem de como os elementos gerados são mostrados corresponde a ordem em que os componentes foram utilizados para representar esses elementos no momento da prototipação no Balsamiq. Por isso em alguns códigos gerados, a ordem em que os elementos foram adicionados no protótipo não corresponde exatamente à ordem que eles deveriam aparecer na tela e isso faz com que haja a necessidade de estruturar manualmente boa parte do código no arquivo HTML gerado. Assim, esse fato de ter que reestruturar o código gerado, fez o Voluntário 1 levar mais tempo para deixar o código no formado em que considera adequado e que geralmente desenvolve e organiza seus projetos.

Já o Voluntário 2 disse que o aprendizado por ter desenvolvido a tela do seu protótipo desde o início sem o uso da ferramenta o ajudou a lembrar uma parte do código, já que o mesmo teve esforço externo para pesquisar e com isso trechos de códigos ainda estavam em sua mente, o que segundo ele, agilizou o desenvolvimento nesta segunda etapa do estudo. Mas ressaltou também que os componentes ou algumas *tags* HTML geradas estavam mal identadas e desordenadas. Por outro lado, entende que é uma boa proposta para ser utilizada na faculdade por alunos, pois pode facilitar o aprendizado da linguagem e o desenvolvimento de projetos.

Entende-se que ambos os voluntários retornaram uma visão diferente sobre a ferramenta. Ambos os voluntários identificaram limitações na ferramenta e essas limitações já são conhecidas uma vez que a ferramenta foi desenvolvida para reconhecer um conjunto de específico de componentes e que essa estrutura de código para cada componente definida num dicionário ia causar inconsistência em algum momento, pois

os componentes do Balsamiq podem ser atualizados e não condizer mais com aquele código. Além de que a cada dia, novas formas de desenvolvimento de interface ou de componente (inclusive com Bootstrap) no *front-end* são criadas, o que torna rapidamente essa estrutura defasada.

Desse modo, tendo em vista que a ferramenta foi testada e que foram obtidos alguns *feedbacks* práticos dos voluntários, verifica-se que a ferramenta pode ser utilizada para fins de treinamento ou de aprendizagem de desenvolvedores iniciantes. Além de que algumas oportunidades de melhorias surgiram através destes *feedbacks*. Assim, entende-se que os resultados obtidos são válidos para concluirmos sobre a utilidade da ferramenta, baseado no problema abordado neste trabalho.

5

CONCLUSÃO

5.1 Considerações Finais

Este trabalho apresentou o desenvolvimento de uma ferramenta com o objetivo de acelerar o tempo de desenvolvimento de *front-end* quando se tem protótipos de telas. Esta ferramenta é do tipo Gerador de Código e para o desenvolvimento desse gerador, nomeado de Proto2Code, foi usada a metodologia descrita em Saturno (2007) que, resumidamente, tinha as etapas de: escrever o código manualmente, seja de componentes, seja de telas, para identificar informações para a estrutura de entrada e saída do gerador; escolher a tecnologia para desenvolver o gerador; desenvolver "o esqueleto"ou a lógica geral desse gerador; desenvolver o processador de entrada; testar e criar modelos de saída e por fim, desenvolver o processador de saída. Depois, a ferramenta foi submetida a uma avaliação através de um estudo piloto com dois voluntários e desenvolvedores da indústria para colher *feedbacks* sobre a viabilidade de uso.

Sendo assim, embora os resultados sejam preliminares e baseado nos resultados da avaliação que foi realizada, o Proto2Code tem potencial de acelerar o desenvolvimento de *front-end* quando os desenvolvedores são iniciantes. Isso porque no caso de desenvolvedores experientes, os padrões de programação e de geração da ferramenta acabam fazendo com que eles percam mais tempo adaptando às suas práticas e modelos de trabalho por já terem um jeito de desenvolver seus projetos, com suas técnicas e suas formas de estruturar o código.

O *script* que compõe a ferramenta em si é de fato um *script* gerador de código porque o resultado da execução dele é a geração de código fonte, uma vez que ele consiste

em manipular dados de um arquivo para então processar e gerar outro arquivo. Este *script* possui limitações, conforme comentado na seção 4.2, que podem ser endereçadas em trabalhos futuros para melhoria da ferramenta.

Portanto, com essa versão do Proto2Code, percebeu-se que o voluntário mais experiente levou mais tempo para deixar o código gerado nos seus padrões de uso do que ter desenvolvido sozinho sem o Proto2Code. Os resultados e *feedbacks* dão indícios de que a ferramenta pode ser utilizada para treinamento de desenvolvedores *front-end* iniciantes e para acelerar o seu processo de desenvolvimento e aprendizado. Assim, a ferramenta pode ser útil para usuários mais inexperientes e principalmente, para alunos que estão aprendendo a desenvolver.

5.2 Trabalhos Futuros

Nesta seção, apresentamos algumas propostas para desenvolvimentos futuros relacionados à ferramenta exposta neste trabalho. Estas propostas podem ser divididas em duas categorias. Na primeira, consideramos modificações que podem ser feitas no script e, na segunda, indicamos potenciais incrementos para a utilização da ferramenta.

Para o *script*, pode-se descrever como melhorias a lógica de programação. Isso porque ele apenas reconhece o conjunto de elementos que estão mapeamos no 'dicionário de componentes'. Caso seja submetido um JSON gerado por outra ferramenta que não seja o Balsamiq ou pelo próprio Balsamiq em outras versões que tenham componentes diferentes, possivelmente o *script* não vai funcionar porque não identificaria tal componente por não estar mapeado ou pronto para se adaptar a essa exigência. Uma observação é: à medida que o Balsamiq for sendo atualizado e também incrementado de mais componentes para uso, o Proto2Code deve ser também atualizado através do código para esses componentes.

Assim, a proposta para isto num trabalho futuro é deixar que os elementos do JSON nos quais a ferramenta é capaz de interpretar seja customizável, sendo possível a inserção de novos componentes à midida que eles forem surgindo no Balsamiq. Além disso, realizar novos estudos com novos voluntários para identificar a potencialidade

da ferramenta. Baseado nesse avanço, tem-se algumas sugestões para a implementação na ferramenta.

Uma delas é no modo de uso. Hoje, para usarmos o Proto2Code, é necessário fazer tudo no código fonte, ou seja, no *script* principal, bem como adicionar o caminho do arquivo JSON na máquina local do usuário, adicionar o caminho de saída também na máquina local do usuário e escolher o nome do arquivo HTML que será gerado. Então um trabalho futuro seria criar uma interface gráfica, ou uma API que integrasse essa interface com o *script* em Python, em que o usuário pudesse apenas fazer o upload do seu arquivo JSON e escolher o caminho de saída onde será salvo o arquivo HTML e através de um botão de “Gerar”, teria rapidamente o arquivo disponível para uso, sem ter que acessar o código fonte, no que ficaria trabalhando como uma “caixa preta”.

Uma outra sugestão é possibilitar a geração de código *front-end* nas estruturas de interfaces estilizadas baseadas em diferentes *frameworks* de desenvolvimento da atualidade, como por exemplo Vue JS, Angular JS e React JS, e não ficar apenas no modo tradicional como HTML, CSS e Bootstrap.

Outra sugestão é desenvolver um novo gerador, porém dessa vez visando o *Back-End*. Aqui pode-se dizer que é "dois em um", pois uma vez tendo os dois separados, pode-se integrar ambos os geradores, juntaria-se o Proto2Code com o gerador *Back-End* e teríamos um gerador "Full-Stack".

REFERÊNCIAS

ALÉSSIO, S. C.; SABADIN, N. M.; ZANCHETT, P. S. *Processos de Software*. São Paulo: Centro Universitário Leonardo Da Vinci, 2017. ISBN 9788551500460. 11

ASIROGLU, B. et al. Automatic html code generation from mock-up images using machine learning techniques. IEEE Xplore, Turkey, 2019. 28

ASSUNÇÃO, W.; MENDONÇA, W.; VERGILIO, S. Reúso de software: Do oportunista ao sistemático. In: *Anais da II Escola Regional de Engenharia de Software*. Porto Alegre, RS, Brasil: SBC, 2018. p. 157–162. ISSN 0000-0000. Disponível em: <https://sol.sbc.org.br/index.php/eres/article/view/10069>. 22

Balsamiq Wireframes. 2008. Disponível em: <https://balsamiq.com/wireframes/>. 31

BAUMER, D. et al. User interface prototyping-concepts, tools, and experience. In: IEEE. *Proceedings of IEEE 18th International Conference on Software Engineering*. [S.l.], 1996. p. 532–541. 17, 18

BRITTO, T. et al. Técnicas de prototipação para smartphones no apoio à avaliação de interfaces com o usuário. In: . São Paulo: [s.n.], 2011. 17, 19, 20

CroSoften. *A Importância da Prototipação do Software*. 2021. Disponível em: <https://crosoften.com/a-importancia-da-prototipacao-do-software/>. 11

FERREIRA, H. N. M.; NAVES, T. F. Reuso de software: suas vantagens, técnicas e práticas. In: . Uberlandia - MG: [s.n.], 2011. 12

HERRINGTON, J. *Code Generation in Action*. USA: Manning Publications, 2003. ISBN 1930110979. 14, 25, 26, 34

LOPEZ, M. R. F. *Estudo da prototipação na engenharia de requisitos para desenvolvimento de softwares interativos em ciclo de vida acelerado*. Tese (Doutorado) — Master's thesis, Instituto de Pesquisas Tecnológicas do Estado de São Paulo . . . , 2003. 18

MDN contributors. 2005. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web>. 30

Microsoft. 2015. Disponível em: <https://code.visualstudio.com/docs>. 33

MONTEIRO, C. M.; LIMA, F. B. Gerador de códigos para desenvolvimento de aplicações web a partir da modelagem entidade-relacionamento. E-xacta, Belo Horizonte., 2016. 24, 26, 27

MOREIRA, E.; MRACK, P. Sistemas dinâmicos baseados em metamodelos. II Workshop de Computação e Gestão da Informação., 2003. 23

MULINARI, B. *Pandas Python: vantagens e como começar*. 2021. Disponível em: <https://harve.com.br/blog/programacao-python-blog/pandas-python-vantagens-e-como-comecar/>. 36

NIRANJAN, P.; RAO, C. V. G. A mock- up tool for software component reuse repository. International Journal of Software Engineering & Application (IJSEA), India, 2010. Disponível em: <https://airccse.org/journal/ijsea/papers/0410ijsea1.pdf>. 21, 22

OTTO, M.; THORNTON, J. 2011. Disponível em: <https://getbootstrap.com.br/>. 31

Pandas. 2009. Disponível em: <https://pandas.pydata.org/about/index.html>. 32

PRESSMAN, R.; MAXIM, B. *Engenharia de Software - 8ª Edição*. [s.n.], 2016. ISBN 9788580555349. Disponível em: <https://books.google.com.br/books?id=wexzCwAAQBAJ>. 16

ROSEMBERG, C. et al. Prototipação de software e design participativo: uma experiência do atlântico. In: . [S.l.: s.n.], 2008. p. 312–315. 16, 17, 19, 20

SATURNO, C. E. *PLUGIN PARA O EA: GERAçÃO DE INTERFACES DE USUÁRIO A PARTIR DE UM PROJETO DE TELAS*. Monografia (TCC) — Universidade Regional de Blumenau, Blumenau-SC, 2007. 14, 24, 25, 27, 34, 52

SOMMERVILLE, I. *Engenharia de Software*. 9. ed.. ed. São Paulo: Pearson Prentice Hall, 2011. v. 9. ed. 11, 17, 21, 22, 23

SOUTO, M. 2019. Disponível em: <https://www.alura.com.br/artigos/o-que-e-front-end-e-back-end>. 29

SPAK, F. *Dicionários em Python*. 2019. Disponível em: <https://iaexpert.academy/2019/09/12/dicionarios-em-python/>. 39

WAZLAWICK, R. *Engenharia de software: Conceitos e práticas*. Elsevier Editora Ltda., 2019. ISBN 9788535292732. Disponível em: <https://books.google.com.br/books?id=d1qnDwAAQBAJ>. 18, 20

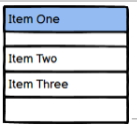
A

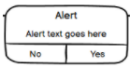
APÊNDICE

JSON	HTML	Platform	Simbolo
TextInput	<input type="text">	Web	
Button	<button type="button" class="btn btn-primary">Primary</button>	Web	
Label	<label> </label>	Web	
SubTitle		Web	A Subtitle
TitleWindow		Web	
Title		Web	A Big Title

JSON	HTML	Platform	Simbolo
Paragraph	`<p> </p>`	Mobile	
Image	`<img src=" " alt="" width=XX height=YY>`	Web	
Webcam	`<video autoplay="true" id="webCam"> <\video>`	Web	

VideoPlayer	`<video width="320" height="240" controls>`
	`  <source src="movie.mp4" type="video/mp4">`
	`  <source src="movie.ogg" type="video/ogg">`
	`  Your browser does not support the video tag.`
	`</video> or <a href=" " title=" "> Name </a>`
		Web	

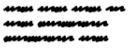
Accordion	`<button class=" ">Section 1</button>`
	`<div class="panel">`
	`  <p>Text 1</p>`
	`</div>`
	
	`<button class=" ">Section 2</button>`
		Web	

Alert	`alert(" ") or confirm(" ");`
	`if (...)`
	`  ....`
	`else..`
		Web	

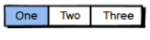
JSON	HTML	Platform	Simbolo

AppBar / Menu	`<div id="mySidebar" class="sidebar">`
	`  <a href="javascript:void(0)" class="closebtn"`
	`onclick="closeNav()">×</a>`
	`  <a href="#">About</a>`
	`  <a href="#">bar2</a>`
	`  <a href="#">bar3</a>`
		Web	

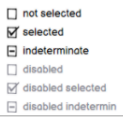
Arrow	`<p>Right arrow: <i class="arrow right"></i></p> or`
	`<p>Left arrow: <i class="arrow left"></i></p> or`
	`<p>Up arrow: <i class="arrow up"></i></p> or`
	`<p>Down arrow: <i class="arrow down"></i></p>`
		Web	

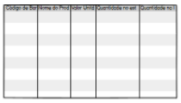
BlockOfText	`<b>Texto</b>`	Web	

BreadCrumbs	`<nav aria-label="breadcrumb">`
	`  <ol class="breadcrumb">`
	`  <li class="breadcrumb-item"><a href="#">Home</a></li>`
	`  <li class="breadcrumb-item"><a href="#">Biblioteca</a></li>`
	`  <li class="breadcrumb-item active"`
	`aria-current="page">Dados</li>`
		Web	

ButtonBar	`<div class="tab">`
	`  <button class="tablinks" onclick="openCity(event,`
	`'One')">One</button>`
	`  <button class="tablinks" onclick="openCity(event,`
	`'Two')">Two</button>`
	`  <button class="tablinks" onclick="openCity(event,`
		Web	

Calendar	`<input id="date" type="date">`	Web	

JSON	HTML	Platform	Simbolo
Callout	 Callout Header × 	Web	
BarChart	 <div class="bar" style="--bar-value:7%;" data-name="Data	Web	
PieChart	não mapeado	.	
LineChart	não mapeado	.	
CheckBox	<input type="checkbox" id="Check" onclick="Function()">	Web	
CheckBoxGroup	<label class="container">One <input type="checkbox" checked="checked"> </label> <label class="container">Two	Web	

JSON	HTML	Platform	Simbolo
CircleButton	<button class="button button5">50%</button>	Web	
ComboBox	 <button class="btn btn-secondary dropdown-toggle" type="button" id="dropdownMenuButton" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false"> Botão dropdown </button>	Web	
StickyNote	não mapeado	.	
DataGrid	<table class="table table-striped"> <thead> <tr> <th scope="col">#</th> <th scope="col">Primeiro</th> <th scope="col">Último</th>	Web	
DateChooser	<input id="date" type="date">	Web	
DataPicker	não mapeado	Mobile	

JSON	HTML	Platform	Simbolo
HRule	`<hr>`	.	
Vrule	não mapeado	.	

HorizontalScrollBar	`<div class="scrollmenu">`
	`  <a href="#home">Home</a>`
	`  <a href="#news">News</a> ...`
	`</div>`
		Web	

Hslider	`<div class="slidecontainer">`
	`  <input type="range" min="1" max="100" value="50"`
	`class="slider" id="myRange">`
	`</div>`
		.	

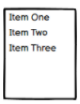
VSplitter	não mapeado	.	

HelpButton	`<link rel="stylesheet"`
	`href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0`
	`/css/font-awesome.min.css">`
	
	`<p>Icon buttons:</p>`
	`<button class="btn"><i class="fa fa-help"></i></button>`
		.	

JSON	HTML	Platform	Simbolo
Icon	`<div id="charging" class="fa"></div>`	.	

IconLabel	`<div>`
	`<i class="fas fa-cloud" style="font-size:36px;"></i>`
	`<label> Nuvem </label>`
	`</div>`
		.	

LineOfText	`<b>Texto</b>`	.	
Link	`<a href="url">Exemplo</a>`	.	a link

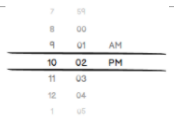
List	`<ul id="myUL">`
	`  <li><a href="#">Item/nome1</a></li>`
	`  <li><a href="#">Item/nome2</a></li>`
	`  <li><a href="#">Item/nome3</a></li>`
			

Menu	`<div class="vertical-menu">`
	`  <a href="#" class="active">Home</a>`
	`  <a href="#">Link 1</a>`
	`  <a href="#">Link 2</a>`
	`  <a href="#">Link 3</a>`
	`  </div>`
		Web	

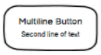
JSON	HTML	Platform	Simbolo
FormattingToolbar	https://www.jegsworks.com/lessons/web/html/fp/formattingbar.htm	Web	

Tooltip	`<div class="tooltip">Hover over me`
	`  <span class="tooltiptext">Tooltip text</span>`
	`</div>`
		Web	

Tree	`<ul id="myUL">`
	`  <li><span class="camada1">Beverages</span>`
	`  <ul class="nested">`
	`  <li>Water</li>`
	`  <li>Coffee</li>`
	`  <li><span class="camada2">Tea</span>`
		Web	

Vcurly	não mapeado	.	

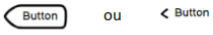
VolumeSlider	`<div class="slidecontainer">`
	`  <input type="range" min="1" max="100" value="50"`
	`class="slider" id="myRange">`
	`</div>`
		.	

MultilineButton	não mapeado	Mobile	

JSON	HTML	Platform	Simbolo
NumericStepper	`<input type="number" id="tentacles" name="tentacles"` `  min="1" max="100">`	Web	

Switch	`<label class="switch">`
	`  <input type="checkbox">`
	`  <span class="slider round"></span>`
	`</label>`
		.	

MediaControls	`<div>`
	`  <button`
	`onclick="document.getElementById('player').anterior">Anterior</`
	`button>`
	`  <button`
	`onclick="document.getElementById('player').play()">Play</button`
		.	

PointyButton	`<a href="#" class="previous">&laquo; Previous</a> or` `<a href="#" class="next">Next &raquo;</a>`	.	

MenuBar	`<div class="menubar">`
	`  <a class="active" href="#home">Home</a>`
	`  <a href="#news">News</a>`
	`  <a href="#contact">Contact</a>`
	`  <a href="#about">About</a>`
	`</div>`
		Web	

iPadPopover	não mapeado	Mobile	

ProgressBar	`<p>Name</p>`
	`<div class="container">`
	`  <div class="progress">80%</div>`
	`</div>`
		Web	