UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Pedro Henrique Brunoro Hoppe

Modelagem MDD de Frameworks SPA: uma Evolução do Método FrameWeb

Vitória, ES

2023

Pedro Henrique Brunoro Hoppe

Modelagem MDD de Frameworks SPA: uma Evolução do Método FrameWeb

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Programa de Pós-Graduação em Informática

Orientador: Prof. Dr. Vítor E. Silva Souza

Vitória, ES

2023

Ficha catalográfica disponibilizada pelo Sistema Integrado de Bibliotecas - SIBI/UFES e elaborada pelo autor

---

Hoppe, Pedro Henrique Brunoro, 1992-
H798m Modelagem MDD de Frameworks SPA: uma Evolução do Método FrameWeb / Pedro Henrique Brunoro Hoppe. - 2023.
89 f. : il.

Orientador: Vítor Estêvão Silva Souza.
Dissertação (Mestrado em Informática) - Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Engenharia de software. 2. Engenharia de software auxiliada por computador. 3. Framework (Arquivo de computador). 4. Software - Desenvolvimento. I. Souza, Vítor Estêvão Silva. II. Universidade Federal do Espírito Santo. Centro Tecnológico. III. Título.

CDU: 004

Modelagem MDD de Frameworks SPA: uma Evolução do Método FrameWeb

Pedro Henrique Brunoro Hoppe

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Mestre em Informática.

Aprovada em 01 de dezembro de 2023.

Prof. Dr. Vítor E. Silva Souza
Orientador

Profª. Drª. Monalessa Perini Barcellos
Membro Interno, participação remota

Prof. Dr. Cidcley Teixeira de Souza
Membro Externo, participação remota

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
Vitória/ES, 01 de dezembro de 2023

Datas e horários baseados em Brasília, Brasil
Sincronizado com o NTP.br e Observatório Nacional (ON) em 06/12/2023 às 15:14:41

documento_defesa_dissertacao_mestrado_folha_aprovacao_pedro_hoppe

Data e Hora de Criação: 06/12/2023 às 07:55:16

Documentos que originaram esse envelope:
- documento_defesa_dissertacao_mestrado_folha_aprovacao_pedro_hoppe.pdf (Arquivo PDF) - 1 página(s)

Hashs únicas referente à esse envelope de documentos

[SHA256]: b59fac4ca1ac5c791dca2d0e08333c22b1bfa562a1d15d81e1af1db0ae4667a3
[SHA512]: 228dd52c1acf42a3e70f4f891c1d7cdabb180c163710cd3cb7ef52424770358919632a79912301d9014fcbcd12bbed3887c8a1c0a3666bd7923f1eb7e5520cf9

Lista de assinaturas solicitadas e associadas à esse envelope

ASSINADO - cidcley@ifce.edu.br
Data/Hora: 06/12/2023 - 15:14:41, IP: 200.129.48.215
[SHA256]: 1d1ee0840ac3757a869a64c85995cc8c4b771d08c5b5122a63980e2cf1f21f6b

ASSINADO - Monalessa Perini Barcellos (monalessa.barcellos@ufes.br)
Data/Hora: 06/12/2023 - 12:57:56, IP: 200.137.66.1
[SHA256]: ef4f5d55873dde35040046480ced548df63ba81122923770d6398a0f9351ba44

ASSINADO - Vítor Estêvão Silva Souza (vitor.souza@ufes.br)
Data/Hora: 06/12/2023 - 08:02:26, IP: 200.137.66.1, Geolocalização: [-20.273265, -40.305891]
[SHA256]: d4b6c42a4d1b703d8e28e1d4b70b6213804f48f57715e4948200dd5e49c32ce8

Histórico de eventos registrados neste envelope

06/12/2023 15:14:41 - Envelope finalizado por cidcley@ifce.edu.br, IP 200.129.48.215
06/12/2023 15:14:41 - Assinatura realizada por cidcley@ifce.edu.br, IP 200.129.48.215
06/12/2023 12:57:56 - Assinatura realizada por monalessa.barcellos@ufes.br, IP 200.137.66.1
06/12/2023 08:02:26 - Assinatura realizada por vitor.souza@ufes.br, IP 200.137.66.1
06/12/2023 08:02:23 - Envelope visualizado por vitor.souza@ufes.br, IP 200.137.66.1
06/12/2023 07:56:46 - Envelope registrado na Blockchain por vitor.souza@ufes.br, IP 200.137.66.1
06/12/2023 07:56:42 - Envelope encaminhado para assinaturas por vitor.souza@ufes.br, IP 200.137.66.1
06/12/2023 07:55:16 - Envelope criado por vitor.souza@ufes.br, IP 200.137.66.1

Agradecimentos

Primeiramente quero agradecer a Deus, todo-poderoso criador dos céus e da terra, por ter me abençoado e ajudado em todo o processo do mestrado e ter me concedido esta oportunidade.

Aos meus pais, Pedrinho e Izabel, que sou muito grato por terem me gerado e criado e me sinto um privilegiado e abençoado por ser filho deles. Aos meus irmãos, Stella e Saulo. Obrigado amada família por todo apoio que sempre me deram e continuam a dar e em especial durante esse processo do mestrado. Amo vocês.

Ao professor doutor Vítor E. Silva Souza por todo o processo de orientação e apoio que com muita dedicação e profissionalismo esteve me acompanhando durante todo o processo e também sem o qual este trabalho não seria possível. Agradeço também, professor, a todos os ensinamentos a mim ensinados neste período, aprendi muito.

E a todos os amigos e familiares que estiverem orando, torcendo e me apoiando neste período, o meu muito obrigado a todos.

"Não tenho vergonha de confessar que sou
ignorante em relação ao que não sei."
(Marco Túlio Cícero, Advogado e Estadista Romano)

Resumo

No campo da Engenharia Web, muitos métodos foram propostos para guiar desenvolvedores no projeto e codificação de aplicações Web. O método FrameWeb é uma abordagem orientada a modelos que visa o desenvolvimento de sistemas que utilizam determinadas categorias de *frameworks* em sua arquitetura, propondo a utilização de modelos que incorporam conceitos desses *frameworks* durante o projeto. Até o presente momento, o método FrameWeb não considera os *frameworks* do tipo SPA (em inglês, *Single Page Application*) e, nos últimos anos, esse tipo de *framework* ganhou muita popularidade entre os desenvolvedores. Neste trabalho nós propomos adicionar suporte para *frameworks* SPA ao FrameWeb. Este trabalho foi conduzido no contexto da Teoria do Design, em particular aplicando o método *Design Science Research* para propor tal solução. Com a nossa pesquisa, conseguimos atualizar o metamodelo do método FrameWeb para que sua linguagem de modelagem agora suporte estruturas SPA e seus construtos. As ferramentas FrameWeb (editor gráfico e gerador de código) também evoluíram para suportar os novos elementos. Experimentos de modelagem de SPAs existentes com esta nova versão do FrameWeb, gerando código a partir dos modelos e comparando com o original mostraram que, em média, cerca de 69% das *tags* HTML poderiam ser geradas a partir dos modelos. O suporte para *frameworks* SPA no FrameWeb permite que desenvolvedores projetem e modelem suas aplicações utilizando construtos que se relacionam com os *frameworks* utilizados no desenvolvimento, facilitando a comunicação do desenvolvedor por meio dos modelos e gerando código para melhorar a produtividade do desenvolvedor.

Palavras-chaves: MDD. Engenharia de Software. WIS Frameworks. DSL. FrameWeb. Reuso. Engenharia Web. Método. Linguagem. Ferramentas. SPA Frameworks.

Abstract

In the field of Web Engineering, many methods have been proposed to guide developers in designing and coding Web applications. The FrameWeb method is a model-driven approach that targets the development of systems that use certain kinds of frameworks in their architecture, proposing the use of models that incorporate concepts from these frameworks during design. For the time being the FrameWeb method currently does not consider SPA (Single Page Application) frameworks and, in recent years, such kind of framework has gained a lot of popularity among developers. In this work, we propose to add support for SPA frameworks to FrameWeb. This work was conducted under the context of Design Theory, in particular applying the Design Science Research method to propose such solution. With our research, we have managed to updated the FrameWeb meta-model so that its modeling language now supports SPA frameworks and their constructs. FrameWeb tools (graphical editor and code generator) also evolved to support the new elements. Experiments of modeling existing SPAs with this new version of FrameWeb, generating code from the models and comparing with the original showed that, in average, around 69% of the HTML tags could be generated from the models. The support for SPA frameworks in FrameWeb allows developers to design and model their applications using constructs that relate to the frameworks used in the development, facilitating developer communication using the models and generating code to improve developer productivity.

Keywords: MDD. Software Engineering. WIS Frameworks. DSL. FrameWeb. Reuse. Web Engineering. Method. Language. Tools. SPA Frameworks.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

MDD	Model Driven Development
DSL	Domain Specific Languages
GPL	General Purpose Languages
WIS	Web-based Information Systems
SPA	Single Page Aplication
API	Application Programming Interface
REST	Representational State Transfer

Sumário

1 Introdução

Desde o início da Engenharia Web, muitos métodos que auxiliam a construção de aplicativos para a plataforma Web foram propostos. O método FrameWeb é um destes métodos. Ele auxilia no desenvolvimento de sistemas de informação Web, ou WISs (em inglês, *Web-based Information Systems*), que utilizam certas categorias de *frameworks* em sua infraestrutura (e.g., controladores frontais, mapeamento objeto/relacional, injeção de dependência) (SOUZA, 2020).

Para cada categoria de *framework* considerada pelo FrameWeb, o método propõe certos construtos em sua linguagem de modelagem (MARTINS; SOUZA, 2015), utilizados por seu editor gráfico (CAMPOS; SOUZA, 2017) e gerador de código (ALMEIDA; CAMPOS; SOUZA, 2017) para prover um esqueleto do WIS a partir de modelos baseados na UML, seguindo uma arquitetura padrão proposta pelo método, que incorpora as arquiteturas propostas por tais *frameworks*. O método visa auxiliar na comunicação entre os desenvolvedores, documentação da arquitetura do sistema e aumento da produtividade (e.g., por meio de geração de código).

Uma categoria de *frameworks* que se tornou estado-da-prática é a dos *frameworks* SPA (*Single Page Aplication* ou Aplicações de Página Única). Tais *frameworks* auxiliam na construção de aplicações Web cujas funcionalidades estão concentradas em uma única página ao invés de carregar uma nova página a cada operação. Gmail e Facebook são exemplos populares de aplicações SPA. Estes *frameworks* se concentram na parte conhecida como *front-end* da aplicação Web, ou seja, as páginas Web apresentadas como interface com o usuário (que executam do lado do cliente), solicitando serviços do *back-end* (lógica de negócio que executa do lado do servidor), por meio de chamadas HTTP, comumente seguindo o modelo de arquitetura REST (*Representational State Transfer*, ou Transferência Representacional de Estado).

Até o momento, o método FrameWeb não considerou a existência desta nova categoria de *frameworks* (SOUZA, 2020). Isso possivelmente limita a aplicação do método, pois no caso do time de desenvolvimento escolher um *framework* SPA para implementação de um projeto de software — uma escolha cada vez mais comum, devido às vantagens deste tipo de *framework*, cf. Seção 2.4 —, provavelmente não conseguirá representar os elementos arquiteturais desta categoria de *frameworks* nos modelos propostos pelo método, abandonando seu uso ou tentando adaptá-lo com soluções improvisadas e sem suporte de suas ferramentas.

FrameWeb é uma aplicação dos conceitos de desenvolvimento orientado a modelos (em inglês, *MDD – Model Driven Development*), cf. Seção 2.1. Sendo assim, ele visa dar

suporte a desenvolvedores com a intenção de melhorar o desenvolvimento de *software*, facilitando a comunicação da equipe e aumentando a qualidade dos resultados. Além disso, no atual estado-da-arte, não temos uma solução semelhante ao método FrameWeb em que um sistema WIS que utiliza um *framework* do tipo SPA é modelado de forma completa utilizando as técnicas de MDD (cf. Seção 2.5). Tudo isso demonstra a necessidade da adição desta categoria de *frameworks* ao método FrameWeb, contribuindo assim para o estado-da-arte e o estado-da-prática.

1.1 Objetivos

Neste contexto, este trabalho tem por objetivo propor uma abordagem para desenvolvimento de WISs do tipo SPA seguindo o paradigma do MDD por meio da evolução do método FrameWeb. Este objetivo pode ser subdividido nos seguintes objetivos específicos:

- Levantar os elementos arquiteturais desta categoria de *frameworks*;
- Analisar o nível de suporte que o método FrameWeb já proporciona a alguns destes elementos;
- Propor uma evolução do método (novos construtos e ferramentas) de modo que possa dar suporte a *frameworks* SPA.

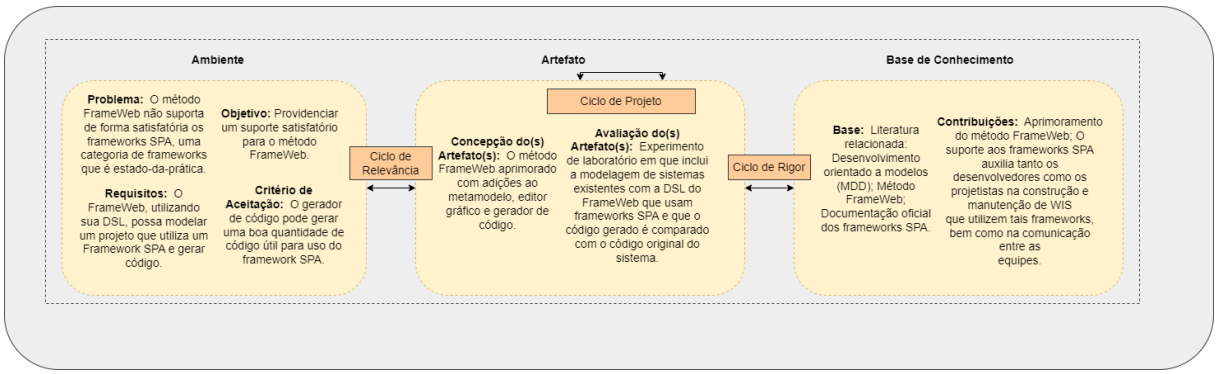
1.2 Método

No desenvolvimento da pesquisa de Engenharia de Software é muito comum ser um desafio demonstrar com sua pesquisa as contribuições que ela proporciona e destacar aos praticantes dela os ganhos da mesma. Isso ocorre parcialmente devido à natureza interdisciplinar da pesquisa nesta área, que foi construída em e pode contribuir para vários campos da ciência, como matemática, engenharia, *design* e ciências sociais. Essas disciplinas se apoiam em diferentes paradigmas de pesquisas e geralmente possuem visões contraditórias entre si sobre o que é considerado uma pesquisa válida e confiável. Além disso, como a Engenharia de Software é um campo profissional também, os praticantes da mesma e outros pesquisadores da área questionam a relevância dos problemas investigados e o valor das soluções propostas (STOREY et al., 2017).

Para solucionar este problema que ocorre também em outras áreas de pesquisa, a *Design Science* é utilizada como arcabouço para auxiliar pesquisadores em suas pesquisas de modo que possam destacar os problemas que estão sendo "atacados" e os benefícios das soluções propostas. Várias técnicas de *Design Science* foram propostas na literatura (STOREY et al., 2017) como, por exemplo as de Storey et al. (2017), Hevner (2007) e Wieringa (2014).

Utilizamos, então, o arcabouço da *Design Science* para definir o método do trabalho, como mostra a Figura 1. Esta descreve a visão geral do nosso trabalho baseado nos princípios do *Design Science*. Na parte de ambiente, temos a definição do problema que queremos resolver, o objetivo a ser alcançado, os requisitos para atender a solução desse problema e o critério de aceitação utilizado para se avaliar se o objetivo foi alcançando. Na parte de artefato, temos a concepção dos artefatos produzidos para a resolução do problema e a avaliação destes mesmos artefatos. As partes de Ambiente e Artefato compreendem o ciclo de relevância do problema, em que é identificado o problema a ser atacado e os artefatos produzidos para a solução do mesmo. Na parte de artefato temos o ciclo do projeto em que os artefatos são construídos e avaliados. Por fim, temos a Base de Conhecimento que define a Base Conceitual para a construção dos artefatos e entendimento do problema, como também a especificação das contribuições que a pesquisa fará. Essa parte junto com a parte de artefato compreende o Ciclo de Rigor.

Figura 1 – Visão geral do método utilizado neste trabalho, baseado no arcabouço da *Design Science*.



Fonte: Produzido pelo autor.

Inicialmente para entendermos o problema era necessário conhecer quais eram as principais características intrínsecas dos *frameworks* SPA. Isso nos direcionou a um estudo de caso exploratório com a codificação de pequenos projetos utilizando os três *frameworks* SPA mais populares (detalhes descritos no Capítulo 3) e também o estudo de suas documentações oficiais cujo o propósito é a investigação do problema.

Tendo o conhecimento das características dos *frameworks* SPA, iniciamos um novo passo em nossa pesquisa visando saber se algum método na literatura possuía suporte satisfatório para a modelagem de WIS que utilizam esta categoria de *frameworks*. Foi realizada uma revisão da literatura para conhecer soluções ou técnicas de MDD propostos para essa categoria de aplicação. Após pesquisar na literatura científica, descobrimos que existem trabalhos que utilizam a abordagem MDD para desenvolvimento de aplicações com *frameworks* SPA, porém focam em partes da arquitetura do projeto ao invés de um sistema

completo como é a proposta do método FrameWeb (o Capítulo 2 traz mais detalhes).

Na sequência, focamos no método FrameWeb, analisando se, até aquele momento, ele conseguiria prover suporte ao desenvolvimento com MDD de WISs do tipo SPA. Após essa iteração, concluímos que o método FrameWeb não possuía, de forma satisfatória, um suporte para *frameworks* SPA. Desta forma, iniciamos um estudo do método FrameWeb para descobrir quais seriam as adições necessárias para que o método passasse a suportar de forma satisfatória essa categoria.

Levantadas as melhorias necessárias, elas foram devidamente implementadas no método e a próxima etapa (propósito de avaliação do artefato) foi direcionada para avaliar se tais melhorias de fato dão suporte aos *frameworks* SPA. Nesta avaliação (descrita em detalhes no Capítulo 4), obtivemos um resultado satisfatório com um valor médio de 69% de *tags* HTML gerados automaticamente a partir da modelagem de WIS do tipo SPA com o método FrameWeb após a evolução proposta.

1.3 Contribuições

As principais contribuições deste trabalho são:

a) Um estudo dos *frameworks* SPA para identificação de seus elementos arquiteturais mais relevantes no contexto do desenvolvimento de WIS;

b) A aplicação do paradigma MDD para o desenvolvimento de WIS do tipo SPA, com a inclusão de tais elementos arquiteturais nos modelos de projeto e a subsequente geração de código a partir dos modelos, no contexto do método FrameWeb;

c) A evolução do método FrameWeb, com a inclusão destes elementos arquiteturais ao metamodelo que define sua linguagem de modelagem, a inclusão de suporte aos *frameworks* SPA em seu editor gráfico e gerador de código, incluindo novos *templates* para a geração de código para os principais *frameworks* SPA atuais.

1.4 Organização da Dissertação

Além desta introdução, este texto é composto pelos seguintes capítulos:

- O Capítulo 2 apresenta a revisão da literatura com o referencial teórico para este trabalho e os trabalhos relacionados a este;
- O Capítulo 3 detalha as propostas deste trabalho e suas implementações no contexto do método FrameWeb;

- O Capítulo 4 relata de forma detalhada os experimentos realizados para avaliar a proposta de melhoria;
- Por fim, o Capítulo 5 mostra as conclusões deste trabalho.

2 Revisão da Literatura

Este capítulo apresenta os principais conceitos referentes a este trabalho formando assim a base para o entendimento do mesmo. A Seção 2.1 apresenta os conceitos relacionados ao Desenvolvimento Orientado a Modelos, no qual o método FrameWeb é baseado. A Seção 2.2 apresenta os coinceitos de Linguagens Específicas de Domínio enquanto a Seção 2.3 apresenta o método FrameWeb. Já a Seção 2.4 apresenta os *frameworks* SPA. Ao final na Seção 2.5 são apresentados os trabalhos relacionados à proposta desta dissertação.

2.1 Desenvolvimento Orientado a Modelos

O desenvolvimento orientado a modelos (em inglês, *MDD – Model Driven Development*) é uma prática de desenvolvimento de *software* em que, ao invés de o desenvolvedor especificar tudo que um determinado programa deve fazer por meio de uma linguagem de programação, permite aos desenvolvedores modelarem as funcionalidades necessárias e a arquitetura geral que o sistema deve ter (ATKINSON; KUHNE, 2003) visando diminuir o *gap* entre o planejamento (i.e., a arquitetura) e o desenvolvimento de *software* (i.e., a codificação) (PASTOR; MOLINA, 2007).

Ao invés de os desenvolvedores especificar todo detalhe que um sistema deve ter por meio de uma linguagem de programação, o MDD permite que eles modelem quais funcionalidades são necessárias e qual arquitetura geral o sistema deve ter. Ao aumentar os níveis de abstração, o MDD visa automatizar muitas das complexas (mas rotineiras) tarefas de programação (como, por exemplo, fornecer suporte para persistência do sistema, interoperabilidade e distribuição) que ainda têm sido feito manualmente hoje (ATKINSON; KUHNE, 2003).

A característica definidora do MDD é que o foco principal e os produtos do desenvolvimento de *software* são modelos, e não os códigos em si. Obviamente, se os modelos acabam meramente como documentação, eles são de valor limitado, porque documentação diverge muito facilmente da realidade. Consequentemente, uma premissa chave por trás do MDD é os programas serem gerados automaticamente de seus modelos correspondentes (SELIC, 2003), remetendo novamente à ideia de diminuir a distância entre a arquitetura e a codificação do projeto. Normalmente no desenvolvimento de *software*, modelos são utilizados apenas para descrever os elementos do projeto que depois serão codificados em uma linguagem de programação. O MDD, no que lhe concerne, preconiza que os modelos conceituais são usados para "diagramar" ou modelar as funcionalidades e recursos desejados do sistema e, posteriormente, por meio de transformações entre modelos dos diversos níveis de abstração obtém-se o *software* final (MARTINS, 2016; ALMEIDA,

2006).

As principais características do MDD são (MARTINS, 2016):

a) O modelo **é** o projeto do sistema;

b) O modelo é ampliado, evolui e atualiza-se;

c) Há um fluxo contínuo entre as transformações efetuadas nos modelos;

d) A implementação é diretamente derivada do modelo.

Vários autores apresentam vantagens do uso do MDD (KLEPPE et al., 2003; DEURSEN; KLINT, 1998; TRASK; ROMAN, 2006; MERNIK; HEERING; SLOANE, 2005) sendo essa vantagens sumarizadas por Martins (2016): **Produtividade** por ocorrer um aproveitamento melhor do tempo, já que o tempo é gasto na produção dos modelos de alto nível. A **Portabilidade**, pois os modelos de alto nível podem ser transformados em códigos diferentes dependendo da plataforma escolhida. A **Manutenção**, assim também como a **Comunicação**, pois os modelos são parte do *software* e as alterações são realizadas diretamente neles o que facilita a manutenção dos mesmos e também por se tratarem de modelos, facilita no processo de comunicação das partes interessadas. **Otimização**, pois os modelos fornecem recursos que permitem aos geradores prover implementações mais eficientes e com menor incidência de erros e a **Corretude**, pois geradores de código não produzem erros acidentais além de facilitarem a identificação de erros conceituais porque ocorrem em um nível de abstração mais alto.

Alguns autores também citam desvantagens do uso MDD (THOMAS, 2004; AMBLER, 2003) também sumarizados por Martins (2016): A **Rigidez** pois, como a maioria da geração de código não sofre atuação por parte dos desenvolvedores, os *softwares* produzidos são mais rígidos. A **Complexidade**, o **Aprendizado** e o **Investimento**, pois o uso de ferramentas de modelagem, transformação e geradores de código aumentam a complexidade do processo além das técnicas de MDD requerem profissionais com habilidades na construção de linguagens e com experiência nas ferramentas utilizadas e com isso, um maior custo inicial para suprir a necessidade de uma infraestrutura própria. E por fim o **Desempenho**, pois mesmo com o uso de otimizações em alto nível de abstração, os geradores podem produzir bastante código desnecessário, que eventualmente pode apresentar desempenho inferior à codificação manual.

O MDD ganhou bastante atenção tanto da indústria quanto da comunidade acadêmica devido ao seu potencial em aumentar a produtividade e qualidade do desenvolvimento de *software* (MARTINS; SOUZA, 2015; ATKINSON; KUHNE, 2003). No processo do MDD, modelos são especificados com uma sintaxe abstrata clara e regras bem definidas para a sua interpretação. Isso facilita o processamento pelas máquinas, o que por consequência permite a criação de ferramentas para modelagem, validação, transformação, etc. (MARTINS; SOUZA, 2015).

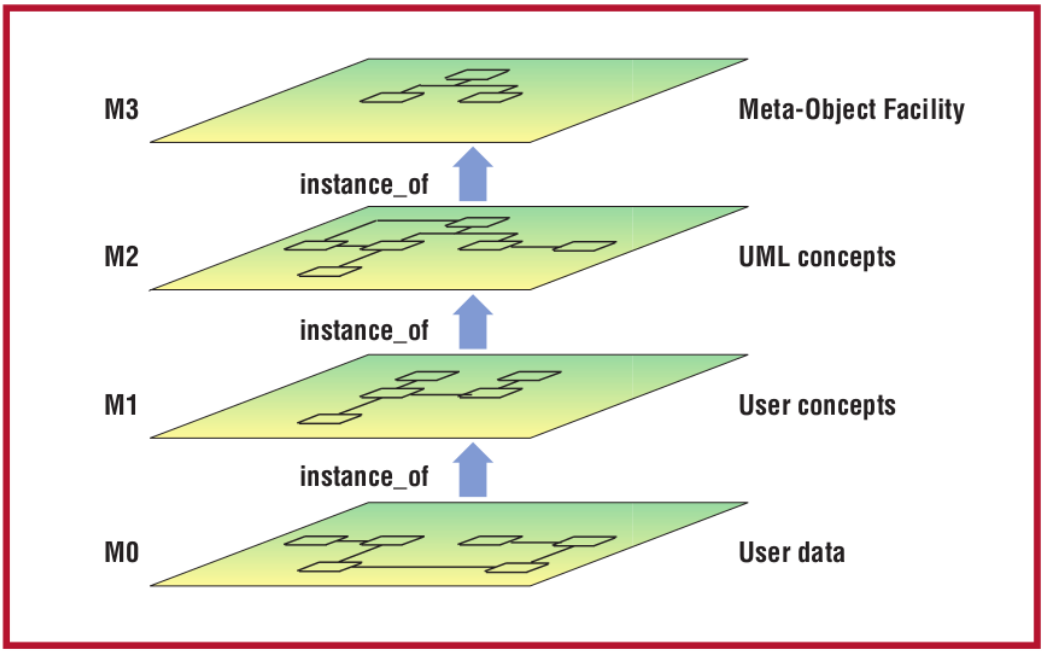


Figura 2 – Infraestrutura tradicional da OMG (ATKINSON; KUHNE, 2003).

2.2 Linguagens Específicas de Domínio

No ano 2000, a OMG,[1] ao percerber os benefícios da prática do MDD, propôs a criação do *framework* MDA (*Model- Driven Architecture*). Conforme o guia MDA da OMG,[2] modelo de sistema é uma descrição ou especificação de sua funcionalidade. Um modelo é geralmente composto por uma combinação de diagramas e textos. Estes podem ser escritos em uma linguagem de modelagem gráfica ou textual (MARTINS, 2016).

A OMG, por meio do MDA exemplifica uma arquitetura tradicional do MDD e propõe uma infraestrutura que consiste em uma hierarquia de níveis de modelos, ilustrada na Figura 2, em que cada um (exceto o topo) é caracterizado como uma instância do nível acima. O nível mais baixo, M0, contém os dados do usuário, i.e., os dados que o *software* foi projetado para manipular. O próximo nível, M1, é dito que contém um modelo dos dados de usuário de M0. Os modelos do usuário residem neste nível. O nível M2 contém um modelo das informações de M1. Por se tratar de um modelo de um modelo, ele é comumente chamado metamodelo (ATKINSON; KUHNE, 2003).

Os modelos de um sistema de informação Web (*Web-based Information Systems* ou WIS) são definidos por uma linguagem gráfica que representa os elementos que compõem esse sistema. Essa linguagem utilizada por si só também é um modelo e, seguindo a ideia de hierarquia da OMG, o modelo que descreve o que cada componente da linguagem representa é o metamodelo da linguagem (ATKINSON; KUHNE, 2003; MARTINS, 2016).

[1] *Object Management Group*, <https://www.omg.org/>
[2] <https://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf>

"No nível do paradigma do MDD, pode-se trabalhar com linguagens para criação de modelos ou metamodelos, execução de transformações e geração de código. Porém, as linguagens podem e devem ser usadas para formalizar domínios. Nesse sentido o MDD pode ser usado na prática para o desenvolvimento de Linguagens Específicas de Domínio (*Domain Specific Languages* – DSL)" (MARTINS, 2016).

Linguagens Específicas de Domínio são linguagens de programação ou linguagens de especificação que visam um domínio de problema específico. Elas não são projetadas para providenciar recursos para resolver todo tipo de problema. Para esse caso usam-se Linguagens de Proposito Geral (*General Purpose Languages* – GPL) como as linguagens C e Java. Porém, por serem específicas para um determinado problema, será mais fácil e rápido utilizar a DSL do que uma GPL para resolver este problema (BETTINI, 2016).

Neste contexto, o método FrameWeb que é um método que aplica o paradigma MDD e para o qual este trabalho propõe melhorias para o suporte a *frameworks* SPA, propõe uma DSL para modelar sistemas Web baseados em *frameworks*. O método FrameWeb será descrito a seguir.

2.3 O Método FrameWeb

Reuso é algo comum no estado da prática do desenvolvimento de *software* como, por exemplo, o uso de bibliotecas, APIs, padrões de projeto e arquiteturais, etc. (FRAKES; KANG, 2005) Uma prática que se destaca neste contexto é o uso de *frameworks* que ajudam a evitar a contínua redescoberta e reinvenção de padrões arquiteturais básicos e componentes, reduzindo o custo e melhorando a qualidade de *software* por meio do uso de arquiteturas e *design* comprovados (SOUZA, 2020; MARTINS; SOUZA, 2015).

Apesar de sua popularidade no uso do desenvolvimento de *software*, até recentemente nenhum dos métodos de Engenharia Web e linguagens de modelagem propostos na literatura considerou a existência de tais *frameworks* antes da fase de codificação do processo de desenvolvimento de *software*. Dado o quanto esses *frameworks* afetam a arquitetura de sistemas Web, isso motivou a proposta do *Framework-based Design Method for Web Engineering*, o FrameWeb (SOUZA, 2020).

O método FrameWeb tem por objetivo auxiliar desenvolvedores na construção de Sistemas de Informação baseados na Web (WISs), tirando proveito da fundação arquitetural oferecida por tais *frameworks* (SOUZA, 2020). Ao longo do seu desenvolvimento, o método vem oferecendo suporte a diferentes categorias de *framework* e atualmente considera as seguintes categorias: controladores frontais, mapeamento objeto/relacional, injeção de dependência e segurança (autenticação e autorização). Tais *frameworks* levaram à definição de uma linguagem de modelagem (MARTINS; SOUZA, 2015) que propõe a construção dos seguintes modelos arquiteturais (SOUZA, 2020):

- **Modelo de Entidades**: responsável por representar as classes do domínio do problema e seus mapeamentos objeto/relacionais, baseados em conceitos de *frameworks* ORM (*Object/Relational Mapping*);
- **Modelo de Persistência**: responsável por representar os DAOs (do padrão arquitetural *Data Access Objects* (ALUR; CRUPI; MALKS, 2003)), que são responsáveis pela persistência das classes de domínio a partir dos seus mapeamentos objeto/relacionais;
- **Modelo de Aplicação**: responsável por representar as classes que implementam as lógicas de negócio do WIS e as relações de dependência entre outras camadas do projeto, utilizando conceitos baseados em *frameworks* DI (*Dependency Injection*);
- **Modelo de Navegação**: responsável por representar os elementos que compõem a camada de apresentação, como páginas HTML, *forms*, etc., além das classes controladoras responsáveis por gerenciar os elementos gráficos e a navegação entre eles (e.g., redirecionamento de páginas). Baseado em conceitos de *frameworks* MVC (*Model-View-Controller*) para a Web (também denominados *frameworks* Controladores Frontais).

O metamodelo do método FrameWeb é baseado no metamodelo UML. A partir disso, o projetista do *software* desenvolve todo o projeto por meio dos modelos mencionados acima, sendo cada um responsável por uma parte do *software*. Após terminada a modelagem, utilizando um editor gráfico (CAMPOS; SOUZA, 2017), o código base do projeto pode ser gerado por meio de um gerador de código (ALMEIDA; CAMPOS; SOUZA, 2017), ambos seguindo a abordagem MDD. O metamodelo que define o FrameWeb é representado na Figura 3 e a arquitetura proposta pelo método é ilustrada na Figura 4.

Devido ao escopo deste trabalho, nos limitaremos a mostrar como exemplo e de forma sucinta o Modelo de Navegação. No exemplo ilustrado pela Figura 5, temos a funcionalidade de cadastro de autores para um sistema de submissão de artigos. As páginas Web são armazenadas no caminho core/registration/, sendo que a página index contém o formulário registrationForm com os componentes inputText e os campos de senha (todos vem da biblioteca *PrimeFaces*,[3] utilizada neste projeto). Assim que o formulário é submetido, o controlador frontal copia os campos do formulário para os atributos de RegistrationController (observe que os campos com o prefixo author. se referem aos atributos do objeto author de RegistrationController ao qual são copiados os respectivos valores do formulário) e o método register() é invocado. Dependendo do resultado do processamento do método, o usuário será encaminhado à página de success ou emailinuse, as quais necessitam que o controlador frontal forneça dados de volta à *view* (author.name e author.email respectivamente).

A Figura 6 mostra um fragmento do metamodelo do Modelo de Navegação do

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

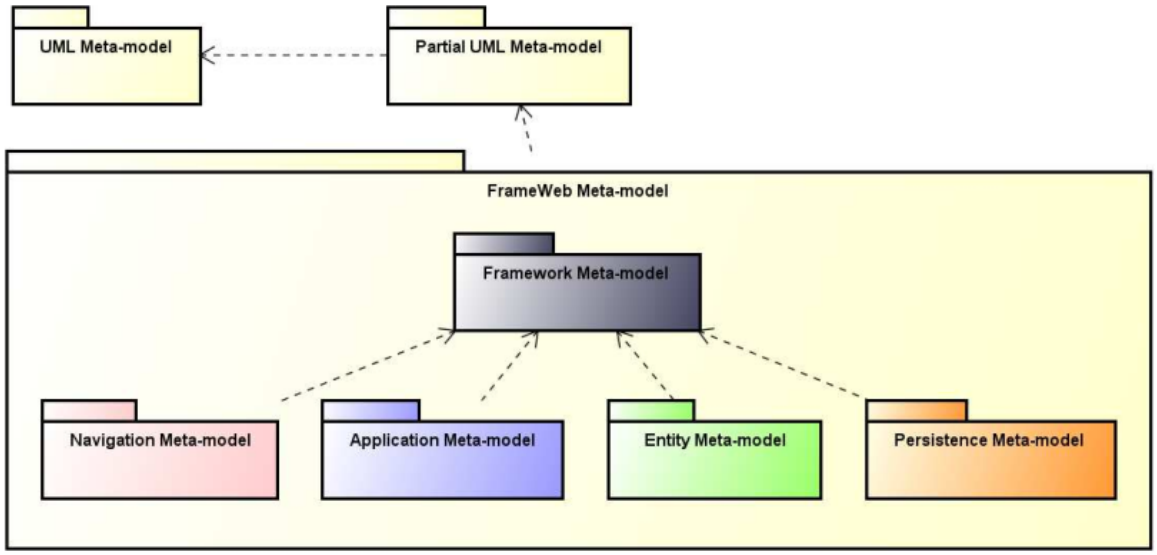


Figura 3 – Metamodelo do método FrameWeb (MARTINS, 2016).

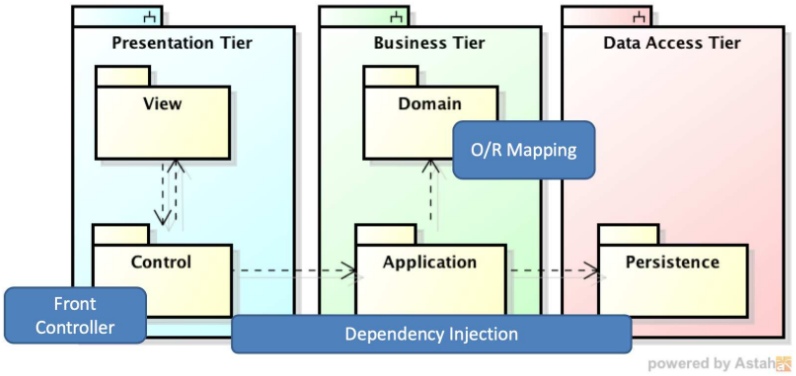


Figura 4 – Arquiteturas representadas pelo método FrameWeb (SOUZA, 2020).

FrameWeb. Os elementos do modelo da Figura 5 são instâncias das metaclasses presentes no metamodelo. Por exemplo, index é instância de Page, RegistrationController é instância de FrontControllerClass e assim por diante. O metamodelo define a linguagem do FrameWeb, restringindo o que pode ou não ser usado nos modelos e guiando a construção de ferramentas como editor gráfico (CAMPOS; SOUZA, 2017) e gerador de código (ALMEIDA; CAMPOS; SOUZA, 2017).

O metamodelo do FrameWeb foi implementado com a ferramenta Eclipse Modeling Framework (EMF)[4] e seu editor gráfico com a ferramenta Sirius,[5] que utiliza o metamodelo como base para definir os elementos gráficos do editor. Para o gerador de código é utilizado o *template engine* JTwig.[6] As implementações dos *templates* são separadas por linguagem

[4] <https://www.eclipse.org/modeling/emf/>
[5] <https://www.eclipse.org/sirius/>
[6] <https://github.com/jtwig/jtwig>

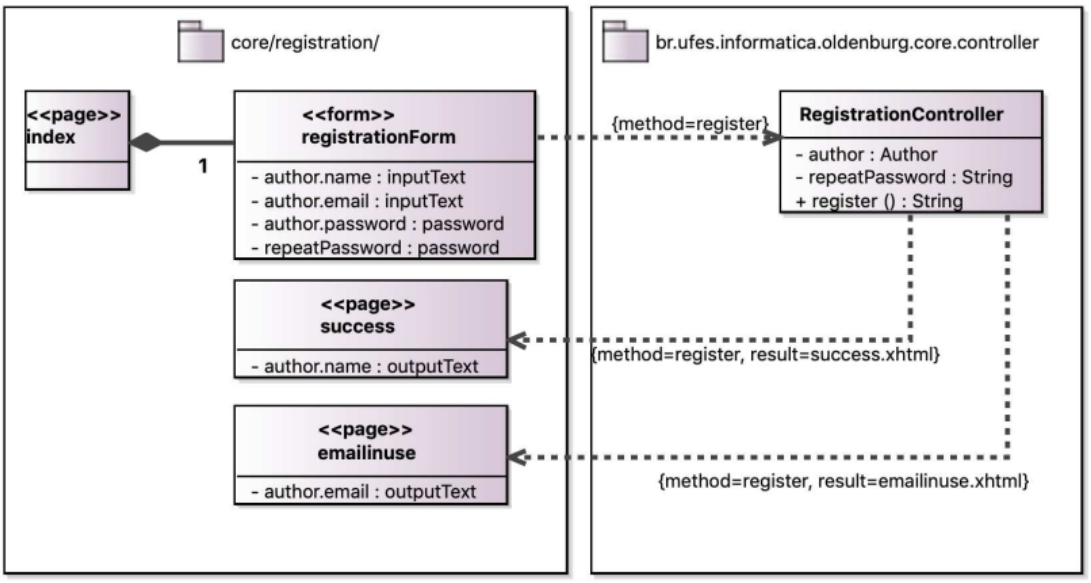


Figura 5 – Exemplo de Modelo de Navegação (SOUZA, 2020).

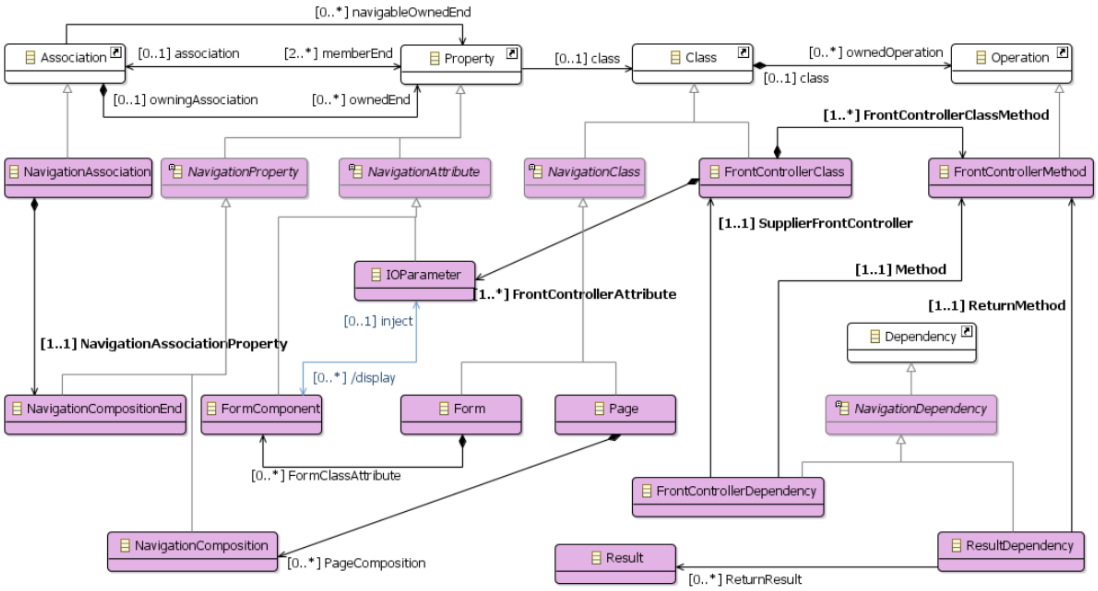


Figura 6 – Fragmento do Metamodelo de Navegação (MARTINS; SOUZA, 2015).

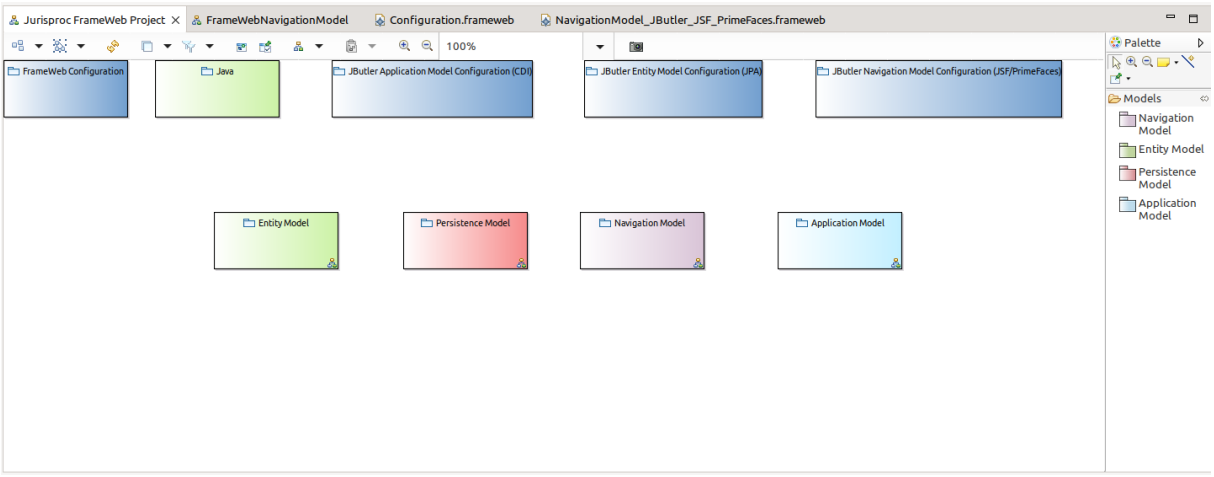


Figura 7 – Versão atual do editor gráfico do método FrameWeb.

de programação e *framework*, assim, conforme preconiza o método FrameWeb, uma linguagem única (DSL) é utilizada para implementar qualquer WIS e a geração de código é que se encarrega de gerar o código base para linguagens e *frameworks* específicos. A Figura 7 mostra a versão atual do editor gráfico. A definição da linguagem e as ferramentas associadas estão disponíveis em um repositório de código-fonte aberto.[7]

2.4 *Frameworks* SPA

Com o objetivo de construir aplicações Web com a aparência de aplicativos *desktop* nativos, várias soluções surgiram, como o IFrames, Java Applets, Adobe Flash e Microsoft Silverlight, sendo adotadas com variados graus de sucesso. Embora diferentes tecnologias, todas elas têm pelo menos um objetivo em comum: trazer o poder de um aplicativo de *desktop* para o ambiente multiplataforma de um navegador da Web. O Aplicativo de Página Única (*Single Page Application* em ingês), ou SPA, compartilha dessa objetivo, mas sem um *plugin* de navegador ou uma nova linguagem de programação para aprender, como em algumas das soluções acima (SCOTT, 2015).

O surgimento das SPAs começou no início dos anos 2000, quando uma nova maneira de pensar sobre o *web-page design* surgiu devido ao uso de requisições assíncronas começar a ganhar popularidade. Primeiro, surge um controlador interessante, porém obscuro, para o navegador Internet Explorer da Microsoft chamado ActiveX, usado para enviar e receber dados de forma assíncrona. Isso eventualmente levou a uma revolução, quando a mesma funcionalidade do controle foi oficialmente adotada pelos principais fornecedores de navegadores com a criação da API chamada *XMLHttpRequest* (XHR). Os desenvolvedores que começaram a mesclar esta API com JavaScript, HTML e CSS obtiveram resultados notáveis (SCOTT, 2015).

[7] <https://github.com/nemo-ufes/FrameWeb>

A combinação dessas técnicas ficou conhecida como AJAX, ou *Asynchronous JavaScript and XML*. As solicitações de dados discretas do AJAX, combinadas com o poder do JavaScript para atualizar dinamicamente o *Document Object Model* (DOM) e o uso de CSS para alterar o estilo da página em tempo real, trouxe o AJAX para a vanguarda do desenvolvimento Web moderno. Pegando carona nesse movimento de sucesso, o conceito SPA propõe elevar o desenvolvimento Web a um nível totalmente novo, expandindo as técnicas de manipulação de nível de página de AJAX para todo o aplicativo. Além disso, os padrões e práticas comumente usado na criação de um SPA pode levar a melhorias gerais no projeto do aplicativo, manutenção do código e tempo de desenvolvimento (SCOTT, 2015).

Em uma aplicação SPA, todo o aplicativo é executado como uma única página da Web. Nesta abordagem, a camada de apresentação para todo o aplicativo é transportada para fora do servidor e gerenciada de dentro do navegador. Para entender melhor a ideia por trás do SPA, vamos comparar com a arquitetura tradicional MVC, ilustrada na Figura 8. Neste modelo de projeto, cada nova requisição de alteração da *view* (páginas HTML) significa uma interação com o servidor. Quando novos dados são necessários na parte do cliente, uma requisição é enviada ao servidor, onde um controlador intercepta a requisição. O controlador então interage com a camada de modelo por meio da camada de serviço, que determina os componentes necessários para concluir a tarefa da camada de modelo. Depois que os dados são buscados, seja por um objeto de acesso a dados (DAO) ou por um agente de serviço, quaisquer alterações necessárias nos dados são feitas pela lógica de negócios na camada de negócios. O controle é passado de volta para a camada de apresentação, onde a visualização apropriada é escolhida. A lógica de apresentação determina como os dados recém-obtidos são representados na exibição selecionada. Frequentemente, a exibição resultante começa como um arquivo de origem com espaços reservados (lacunas, *placeholders*), onde os dados devem ser inseridos (e possivelmente outras instruções de renderização). Esse arquivo atua como uma espécie de *template* de como a exibição é gerada sempre que o controlador encaminha a solicitação para ela. Depois que os dados e a exibição são mesclados, a exibição é retornada ao navegador. O navegador que recebe a nova página HTML e, por meio de uma atualização da interface do usuário, o usuário vê a nova exibição que contém os dados solicitados.

Já na Figura 9 temos um exemplo de arquitetura de uma aplicação SPA baseado na mesma arquitetura da Figura 8 para a camada de modelo. Observe o que aconteceu com a camada de apresentação e as transações externas. Mover o processo de criação e gerenciamento de exibições para a interface do usuário o separa do servidor. A menos que você esteja fazendo uma renderização parcial no servidor, o servidor não precisa mais se envolver na forma como os dados são apresentados, gerando um desacoplamento com a parte do servidor (a parte *back-end*).

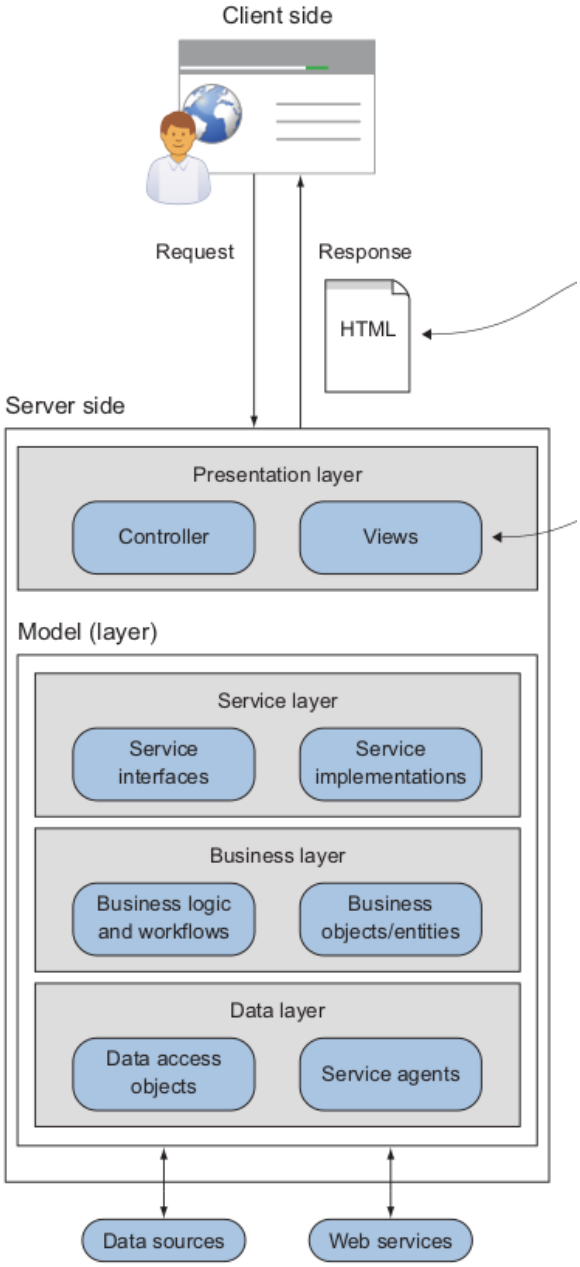


Figura 8 – Arquitetura tradicional MVC (SCOTT, 2015, p. 5).

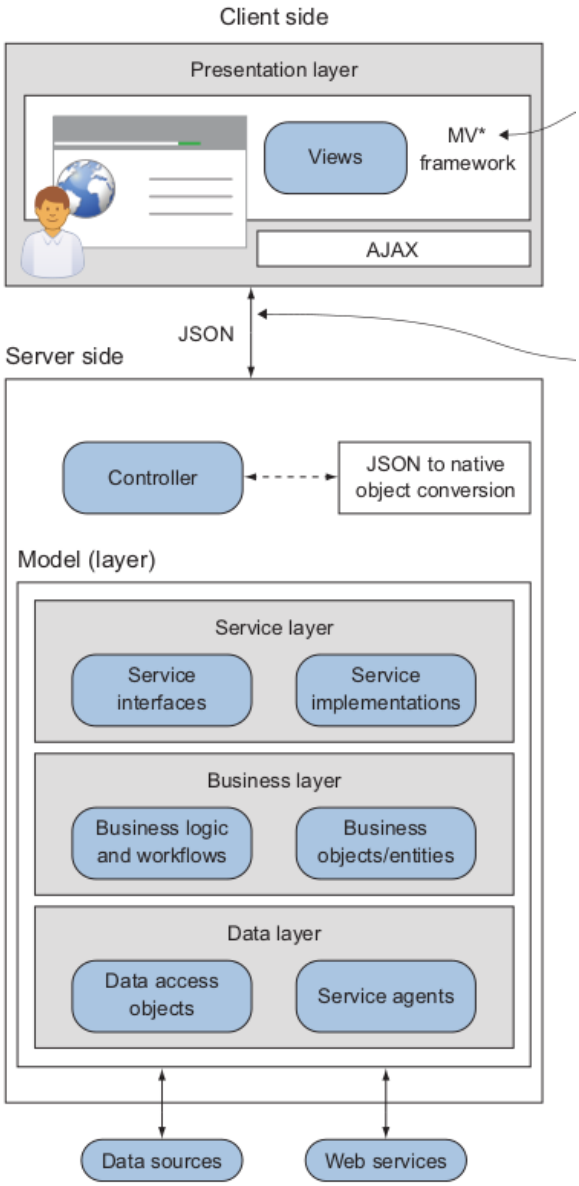


Figura 9 – Aplicação SPA (SCOTT, 2015, p. 6).

As principais diferenças de uma aplicação SPA e um projeto padrão MVC segundo Scott (2015) são:

a) **Sem atualização do navegador:** no SPA, a parte *view* não é composta de páginas HTML estáticas. Elas são apenas partes do DOM que compõem as áreas visíveis da tela. Após o carregamento inicial da página, todas as ferramentas necessárias para criar e exibir a *view* são baixados e preparados para uso no lado do cliente. Se uma nova *view* for necessária, ela é gerada localmente no navegador e anexada dinamicamente ao DOM via JavaScript. Nenhuma atualização do navegador é necessária (como é feito no caso do modelo MVC em que há o carregamento de uma nova página HTML para uma nova *view*);

b) **A lógica de apresentação está no lado do cliente:** como a lógica de

apresentação é feita principalmente no lado do cliente em uma aplicação SPA, a tarefa de combinar HTML e dados é movida do servidor para o navegador. Como no lado do servidor, o HTML de origem contém espaços reservados onde os dados devem ser inseridos (e possivelmente outras instruções de renderização). Este *template* do lado do cliente é usado como base para imprimir novas visualizações no cliente. Porém, não é um modelo HTML para uma página completa. É apenas para a parte da página que a exibição representa;

c) **Transações do servidor:** em uma aplicação SPA, várias abordagens podem ser usadas para renderizar dados do servidor. Esses incluem renderização parcial do lado do servidor, na qual trechos de HTML são combinados com dados na resposta do servidor. O formato da troca de dados é tipicamente JavaScript Object Notation (JSON), embora não seja obrigatório. Isso permite uma melhor portabilidade e flexibilidade do projeto, pois a camada de apresentação fica desacoplada da parte servidor e possibilita a combinação com várias *frameworks* para a camada *back-end* como, por exemplo, usar o *framework* Spring Boot[8] para este fim.

Na Seção 3.1 detalhamos os estudos empíricos feitos com essa categoria de *frameworks* para o entendimento e captação das principais características para a análise das melhorias necessárias ao método FrameWeb para dar suporte a essa categoria.

2.5 Trabalhos Relacionados

Nos últimos anos, foram propostas diversas abordagens baseadas no Desenvolvimento Orientado a Modelos (MDD) de modo a facilitar o processo de desenvolvimento de software em diversos contextos e também com relação a *frameworks* SPA.

Hernandez-Mendez, Scholz e Matthes (2018) propõem uma ferramenta de modelagem de consumo de API's REST no contexto de aplicações SPA. Utilizam-se do arcabouço proposto pelo MDD para desenvolverem um artefato chamado Query Service que é responsável por gerar de forma automática uma "interface" única para consumo de serviços. Este Query Service é definido por um metamodelo construído pelos autores. Eles propõem que os usuários de sua ferramenta modelem sua aplicação com a DSL do Query Service e, após isso, é gerado o esqueleto de código na linguagem específica do *framework* SPA escolhido, cabendo ao desenvolvedor codificar as funções. Este trabalho se concentra apenas em modelar e fornecer uma interface que consome os serviços REST na aplicação SPA, enquanto o método FrameWeb preconiza o desenvolvimento de toda a aplicação e, para as melhorias propostas neste trabalho, a parte *controller* e *view* da parte do cliente.

Outros trabalhos (HAUPT et al., 2014; BONIFÁCIO et al., 2015; ED-DOUIBI

[8] <https://spring.io/projects/spring-boot>

et al., 2016) também propõem modelarem serviços RESTful Web e transformá-los em código, porém eles focam na parte do servidor ao invés do cliente. Deljouyi e Ramsin (2022) propõem um *framework* e um processo para desenvolvimento de serviços Web com a arquitetura REST. Para isso, propõem uma DSL para modelagem de aspectos do serviço e, então, processos de transformações de modelos são utilizados para transformar de um nível de modelo mais abstrato para um mais específico dependente de plataforma. Eles desenvolveram ferramentas também para darem apoio ao método proposto. Este trabalho se concentra apenas em modelar serviços Web REST, não focando em WISs.

Pando e Castillo (2022) propõem uma ferramenta para modelagem de sistemas chamada PlantUMLGen, baseada em uma ferramenta chamada PlantUML, a qual utiliza princípios do MDD para modelar aplicativos MVC e depois o esqueleto de código é gerado para o *framework* Laravel. O objetivo dos autores é incentivar alunos de *design* de *software* a aplicarem técnicas de MDD. Este trabalho se diferencia no nosso na questão de se tratar uma ferramenta de ensino para o incentivo de alunos e não focando em produção industrial, por isso é bem limitada se comparada com a versão atual do método FrameWeb no suporte para o desenvolvimento de WIS. Além disso, não dá suporte a aplicações do tipo SPA, proposta deste trabalho. Já em (ROSSI, 2016) é proposta uma metodologia de desenvolvimento de APIs REST baseada em diagramas UML para modelagem e transformações de modelos para a linguagem de modelagem RAML, específica para modelagem de API's, focando somente nessa parte do projeto e não em um WIS completo, como em FrameWeb.

Ponciano et al. (2020) apresentam uma ferramenta baseada na ideia do MDD chamada EasyContext para desenvolvimento de regras contextuais para dispositivos móveis. Já Sousa et al. (2019) apresentam uma abordagem baseada em MDD para o desenvolvimento de aplicações mulsemedia. Em (MATTOS; MUCHALUAT-SAADE, 2018), também se utiliza a ideia do MDD para mulsemedia. Bezerra e Souza (2019) apresentam uma abordagem MDD para o desenvolvimento de UI (*User Interface*) reativa e personalizável dentro do contexto da Internet das Coisas. Já em (PINTO; GONÇALVES; COSTA, 2019) os autores utilizam a ideia por trás do MDD para apresentarem uma abordagem para gerar *User Interface Prototype* (UIPs) automaticamente a partir de especificações de requisitos Agile escritas em Concordia. Em (OLIVEIRA; SILVEIRA; SOUZA, 2018), os autores propõem uma abordagem MDD para desenvolvimento de sistemas de recomendação.

Dentro do contexto do FrameWeb, Prado e Souza (2018) propõem adicionar suporte aos *frameworks* de segurança por meio de controle de acesso por papel (*role*, em inglês) ao metamodelo, editor e gerador de código FrameWeb. A proposta é bastante similar à trazida neste artigo, porém trata de uma outra categoria de *frameworks*.

Romano e Cunha (2019) propõem a criação de um *framework* chamado *Agile Model*

Driven Development (AMDD) para representar uma "versão *Agile* do MDD" utilizando um novo perfil UML chamado *Web Agile Modeling Language* (Web-AML) e um processo de geração de código a partir dos modelos implementados. Sua diferença com o método FrameWeb é que aquele se foca na aplicação de métodos Ágeis para o desenvolvimento de *software* e também não fornece atualmente todo suporte provido pelo FrameWeb. Moradi, Zamani e Zamanifar (2020) propõem a criação de um *framework* chamado *Context as Service Set of Engeniering Tools* (CaaSSET) para facilitar o desenvolvimento de "serviços de contexto" que seguem a ideia do *Context-as-a-Service* (CaSS), utilizando princípios do MDD. Este trabalho também conta com um metamodelo, modelagem gráfica e gerador de código. No contexto do desenvolvimento de aplicativos *context-aware*, Paspallis (2019) propõe uma metodologia baseada em técnicas de MDD para o desenvolvimento de tais aplicativos com a utilização de *toolchains* e consequentemente geração de código.

Kifouche et al. (2019) propõem a criação de um *framework* chamado *Modesene* (*MOdel Driven Environment for SEnsor NEtwork*) para facilitar o desenvolvimento de uma rede de sensores. *Modesene* foi projetado para assegurar quatro principais processos incluindo modelagem multifacetada, simulação de rede, implantação e exploração de dados. O *framework* proposto suporta ferramentas de redes de sensores por meio do fornecimento de pontes entre essas ferramentas e também conta com gerador de código. Costa-Soria et al. (2013) propõem a criação de um método chamado PRISMA MDD para o desenvolvimento de *softwares* baseados na Arquitetura Orientada a Aspecto (*Aspect-Oriented Software Architectures* – AOSA). O método provê ainda ferramenta gráfica e geração de código C#. Já Sosa-Reyna, Tello-Leal e Lara-Alabazares (2018) propõem uma metodologia de aplicação de princípios de MDD para o desenvolvimento de aplicações do tipo IoT (*Internet of Things* ou Internet das Coisas) baseado em uma arquitetura de serviços (*Service-Oriented Architecture* – SOA), criando uma espécie de "*framework* conceitual". Agüero et al. (2013) propõem uma abordagem de desenvolvimento de sistema pervasivos baseados em agentes (*Agent-Based Pervasive Systems*) utilizando técnicas de MDD e ferramentas para o desenvolvimento. A diferença destes últimos com o método FrameWeb é o nicho de aplicação de técnicas do MDD ser diferente.

Com relação à geração de código provida pelo método FrameWeb, a indústria também possui soluções nessa área. A plataforma Angular possui uma ferramenta CLI (*command-line interface*)[9] que consegue gerar um esqueleto básico de um novo projeto Angular e também consegue gerar esqueletos de componentes e serviços, porém não gera métodos previamente especificados e nem atributos, algo que o FrameWeb faz. O VueJS também possui *Toolchain* chamado "create-vue"[10] que gera apenas o *scaffolding* do projeto e gerencia eventuais dependências que o projeto possa ter (como, por exemplo, adição

[9] Disponível em <https://angular.io/cli>
[10] Disponível em <https://github.com/vuejs/create-vue>

de *plugins*). De forma semelhante o React possui o "create-react-app"[11] que também gera *scaffolding* do projeto e também gerencia eventuais dependências. O ASP.NET Core[12] também possui uma ferramenta de geração de projetos React e Angular baseados respectivamente no "create-react-app" e "Angular CLI" com a adição de fazer a interface com o ASP.NET Core utilizado como *backend*.

A principal diferença dessas ferramentas com o método FrameWeb é que este provê suporte para projeto arquitetural de *software* utilizando técnicas de MDD, possui suporte para múltiplos *frameworks*, tanto MVC quanto SPA (este último proposto neste trabalho), além de a geração de código ser mais específica, gerando assinatura de métodos, de atributos e, no caso dos SPA, das *tags* HTML especificadas no modelo de navegação. FrameWeb provê uma linguagem de modelagem unificada que pode ser adaptada para gerar código para diferentes *frameworks*, sejam eles atuais, legados ou que venham a ser propostos no futuro.

[11] Disponível em <https://github.com/facebook/create-react-app>

[12] Disponível em <https://learn.microsoft.com/pt-br/aspnet/core/client-side/spa/angular> e <https://learn.microsoft.com/pt-br/aspnet/core/client-side/spa/react>

3 Proposta do Trabalho

Neste capítulo detalhamos nossa proposta para adicionar suporte aos *frameworks* SPA ao método FrameWeb. Com a finalidade de explorar o fenômeno estudado, foram aplicados métodos de estudo de literatura sobre *frameworks* SPA, *design* e criação das propostas para FrameWeb a partir deste estudo e experimentos para avaliar tais propostas. Para os experimentos, os dados foram coletados por observação direta sendo realizada análise estatística para traçar as conclusões.

3.1 Estudo dos *Frameworks* SPA

Os *frameworks* SPA (em inglês, *Single Page Application*), são uma categoria de *frameworks* muito popular na indústria. No entanto, o FrameWeb (cf. Capítulo 2), que é um método que se propõe a prover uma linguagem de modelagem e ferramentas para auxiliarem desenvolvedores na construção de Sistemas de Informação Web (WISs), não considera atualmente tal nova categoria de *frameworks* (SOUZA, 2020).

Portanto, foi realizada uma análise não sistemática com três *frameworks* populares dessa categoria, a saber: Angular, React e Vue.js,[1] por meio do desenvolvimento de pequenos projetos com cada um desses *frameworks*, a fim de capturar suas características comuns, i.e., os elementos arquiteturais que fazem parte da estrutura deste tipo de *framework*. Além dos projetos desenvolvidos, também foram estudas as documentações oficiais de cada um dos três frameworks para analisar a arquitetura proposta por eles. Com isso, objetivou-se analisar o nível de suporte que o método FrameWeb já proporciona a alguns destes elementos e propor uma evolução do método (novos construtos e ferramentas) de modo que possa dar suporte a *frameworks* SPA.

As escolhas desses *frameworks* foram baseadas em sua popularidade de uso no mercado, de acordo com a pesquisa de 2021 realizada pelo popular site *StackOverflow*[2] com seus usuários sobre, dentre outras questões, as tecnologias (linguagens de programação, bancos de dados, *frameworks*, etc.) mais utilizadas por eles. A Figura 10 mostra o resultado da pesquisa. Os três *frameworks* SPA escolhidos estão entre os 5 mais utilizados no quesito *Web frameworks*. O motivo de os *frameworks JQuery* e *Express*, respectivamente em segundo e terceiro lugares neste quesito, não terem sido incluídos é o fato de não se encaixarem na definição de *framework* SPA: o primeiro é um utilitário para codificação em JavaScript, enquanto o segundo é um *framework* usado no desenvolvimento do *back-end*

[1] <https://angular.io/>, <https://reactjs.org/>, <https://vuejs.org/>

[2] <https://insights.stackoverflow.com/survey/2021>, o ano de referência é 2021, pois nossa pesquisa foi iniciada neste ano.

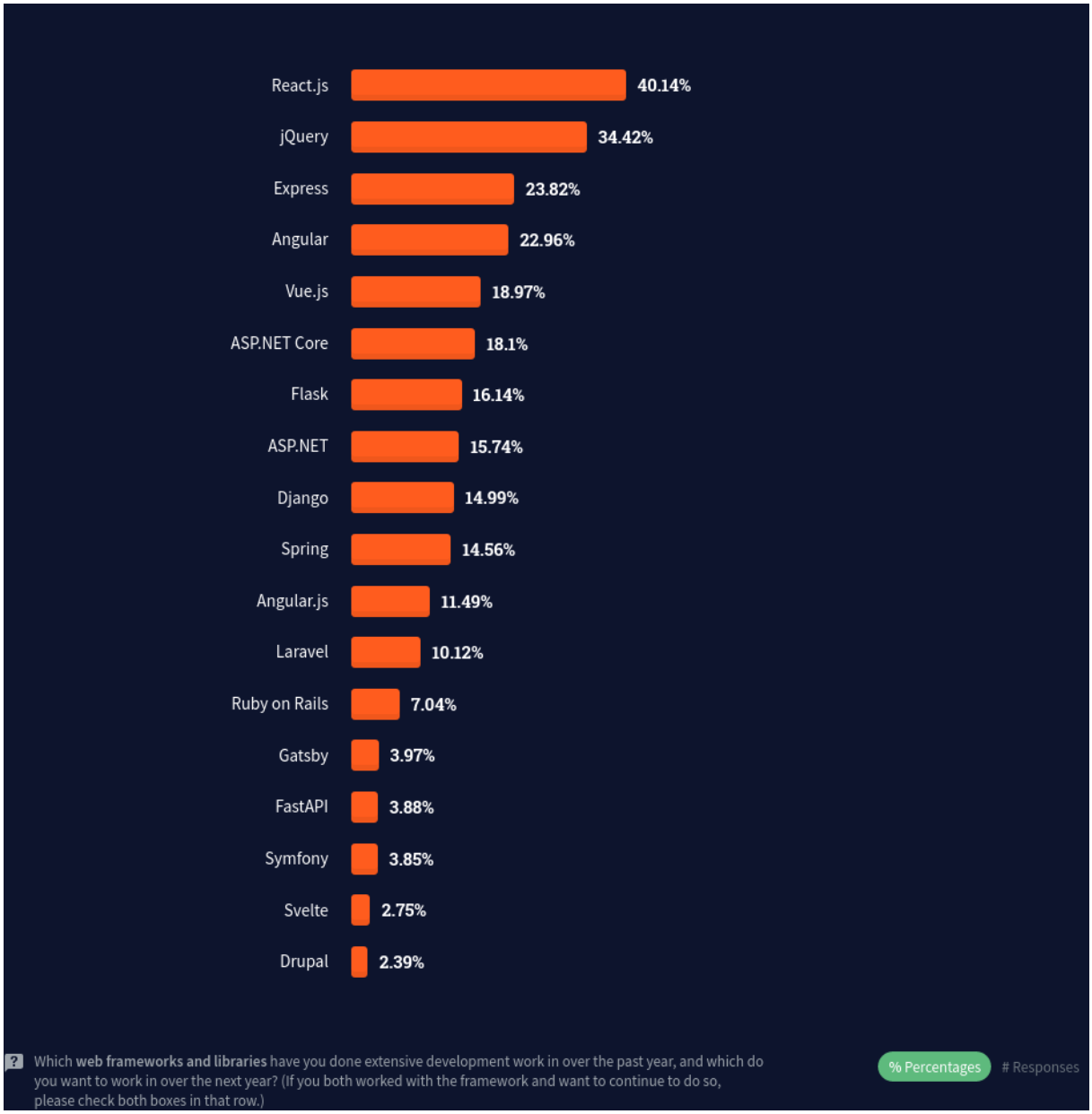


Figura 10 – *Frameworks* mais populares de 2021 de acordo com enquete do site *StackOverflow*

das aplicações.

Para o desenvolvimento dos projetos de cada *framework* escolhido, foi seguida a implementação descrita nos livros *Angular 11 e Firebase* (MACHADO, 2019), *Progressive Web Apps: Construa aplicações progressivas com React* (PONTES, 2018) e *Front-end com Vue.js: Da teoria à prática sem complicações* (VILARINHO, 2017). Cada livro ensina sobre como programar em seus respectivos *frameworks* com os recursos disponíveis para os mesmos de forma didática com a implementação de pequenos projetos de forma assistida. Os projetos, construídos pelo autor desta dissertação, foram:

a) **Angular:** projeto de cadastramento de requisições de departamentos. Disponível em <https://github.com/pedrohbh/estudo-mestrado-angular>;

Figura 11 – Projeto desenvolvido com Angular - Tela de Login.

Fonte: Produzido pelo autor

b) **React:** projeto de uma mini rede social. Disponível em <https://github.com/pedrohbh/react-behappywith.me>;

c) **VueJS:** projeto de sistema similar ao sistema de notas adesivas do Windows. Disponível em <https://github.com/pedrohbh/estudo-mestrado-vue>.

A seguir são apresentados os três projetos implementados para o estudo.

3.1.1 Angular

O projeto desenvolvido com o *framework* Angular consiste em um gerenciador de requisições de departamentos de uma empresa, incluindo um sistema de login e tela de cadastro. As figuras 11, 12, 13, 14 e 15 mostram partes do sistema desenvolvido. A Figura 16 mostra parte do código-fonte do projeto destacando o *component* "Departamento". *Components* são o equivalente aos *Controllers* de *frameworks* MVC tradicionais, como o JSF (neste caso, mais especificamente o *ManagedBean*) mesclado com a camada "view".

Figura 12 – Projeto desenvolvido com Angular - Tela de Login.

Fonte: Produzido pelo autor

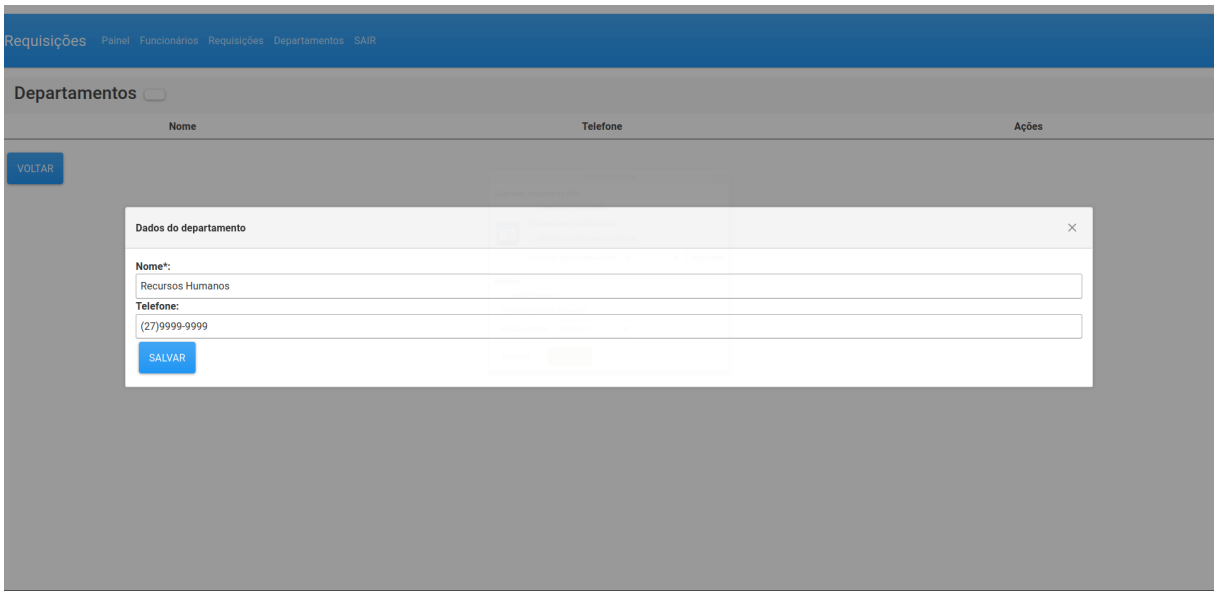


Figura 13 – Projeto desenvolvido com Angular - Tela de Cadastro de Departamento.

Fonte: Produzido pelo autor

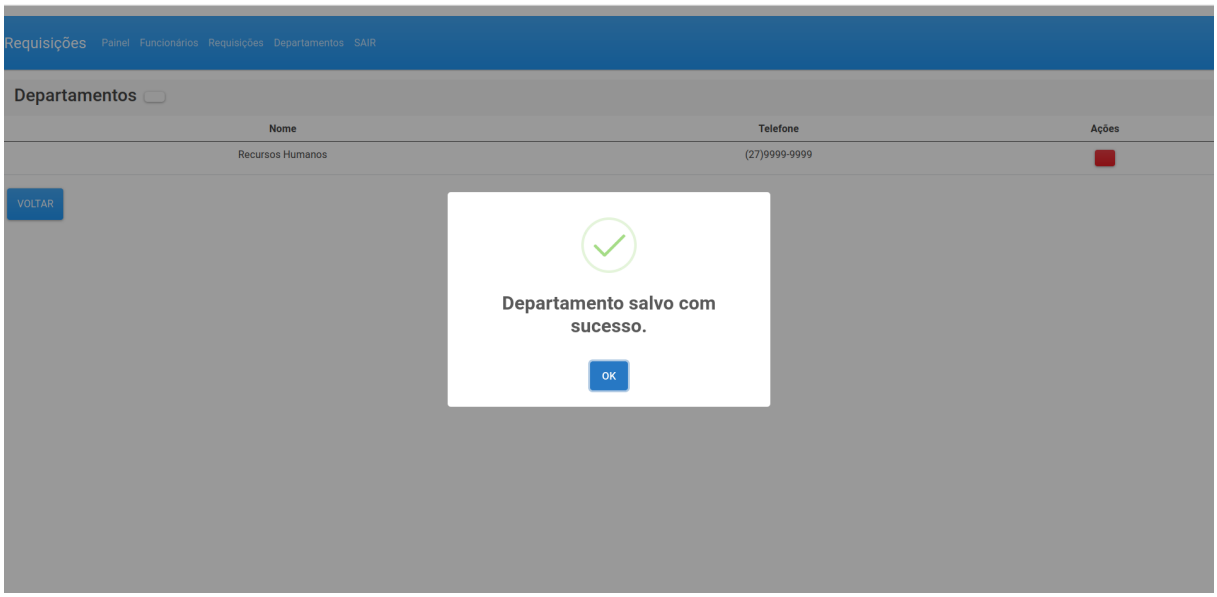


Figura 14 – Projeto desenvolvido com Angular - Tela de Cadastro de Departamento.

Fonte: Produzido pelo autor

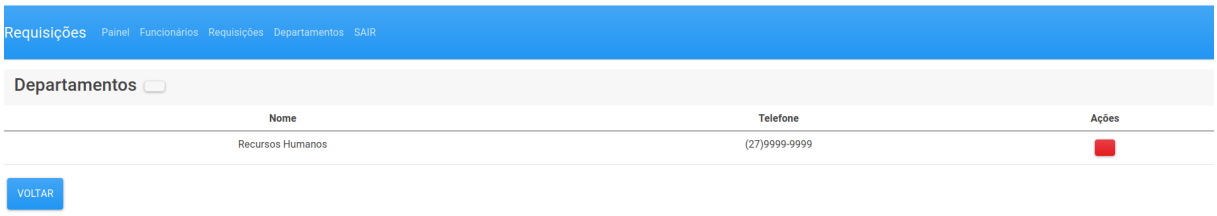


Figura 15 – Projeto desenvolvido com Angular - Tela de Cadastro de Departamento.

Fonte: Produzido pelo autor

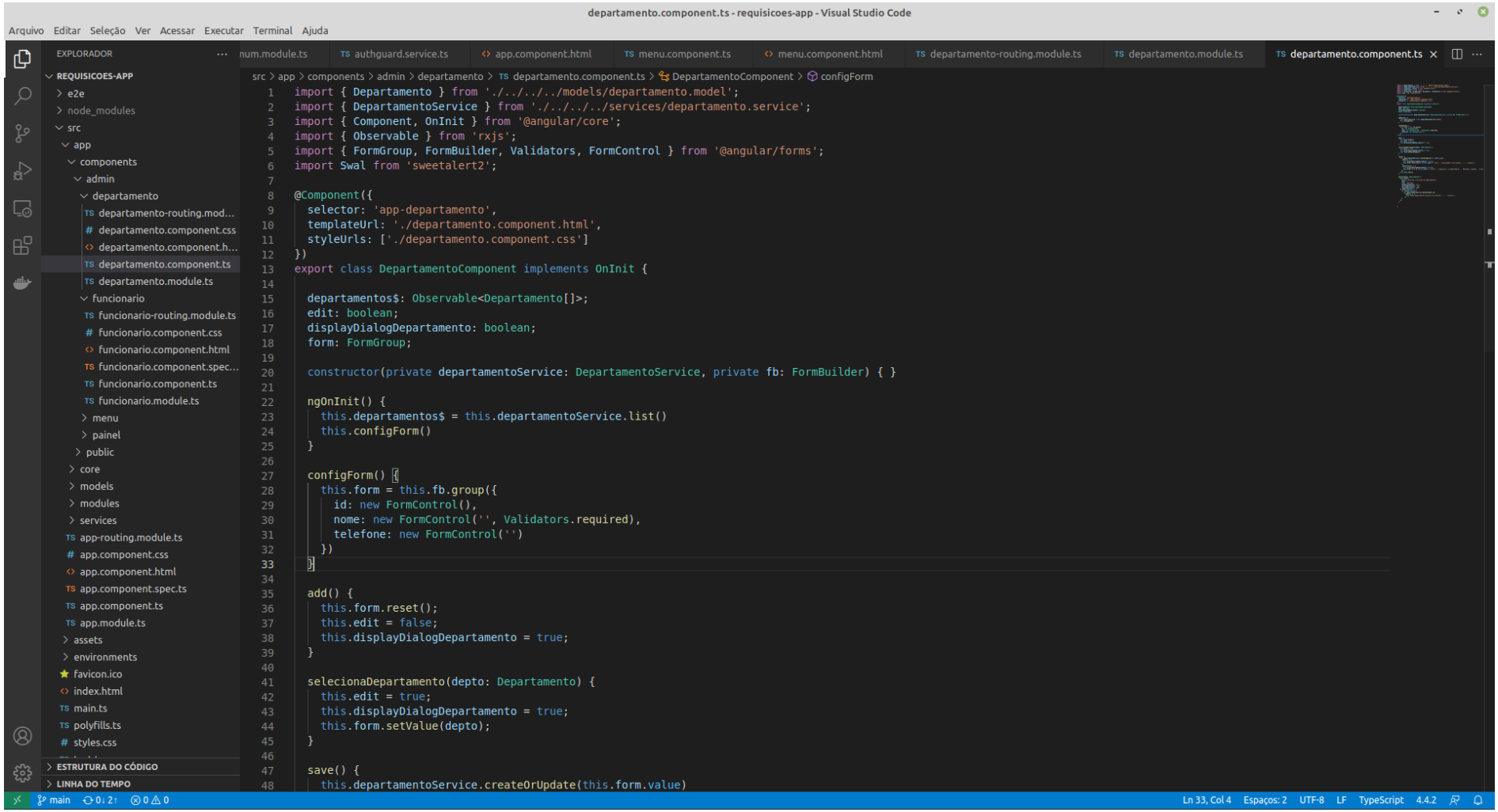


Figura 16 – Código fonte do projeto desenvolvido com Angular com um exemplo de *component*.

Fonte: Produzido pelo autor

3.1.2 React

O projeto desenvolvido com o *framework* React consiste em uma rede social muito simplificada, com cadastro de usuário e login. As figuras 17 e 18 mostram como exemplo o *component* NovoUsuário. As figuras 19, 20, 21 e 22 mostram o projeto desenvolvido. Assim como no caso do Angular, percebeu-se que o React também possui o elemento *component* que desempenha o papel de *controller*, além de estruturar os elementos gráficos da tela por meio da sua função *render* em que é descrito como aquele determinado componente será estruturado por meio do código HTML.

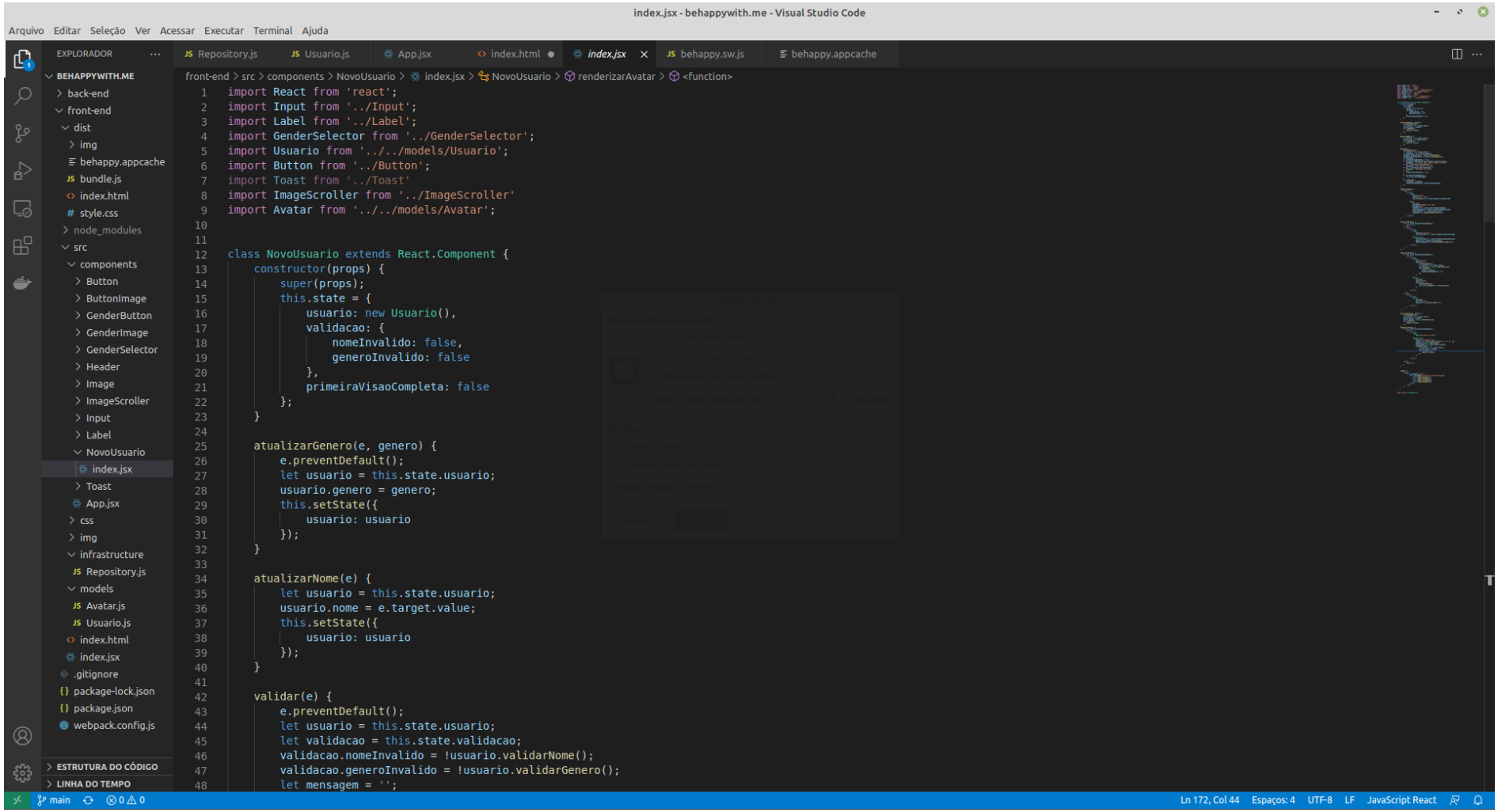


Figura 17 – Código fonte do projeto desenvolvido com React com um exemplo de *component*.

Fonte: Produzido pelo autor

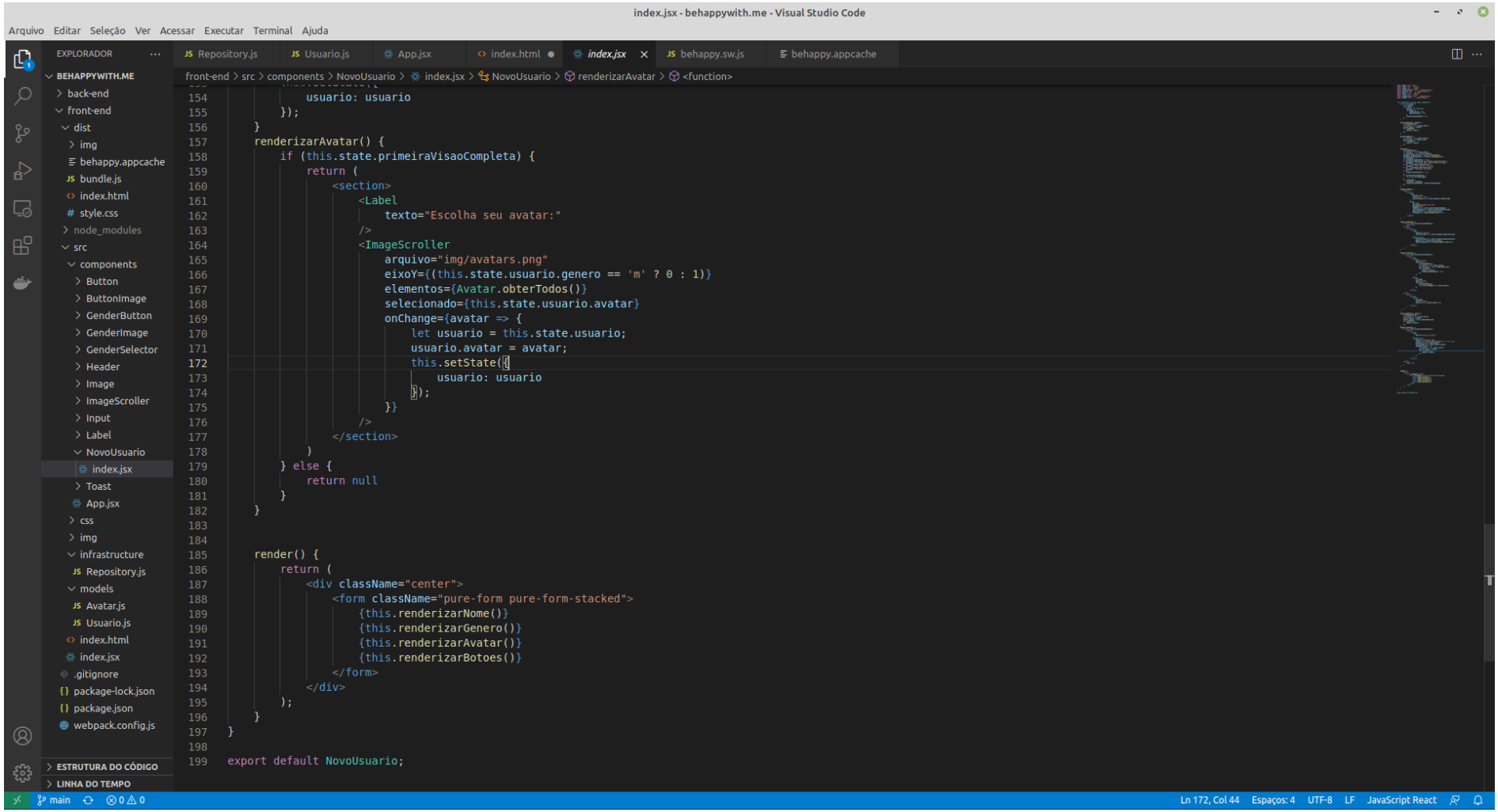


Figura 18 – Código fonte do projeto desenvolvido com React destacando o *render* do *component*.

Fonte: Produzido pelo autor

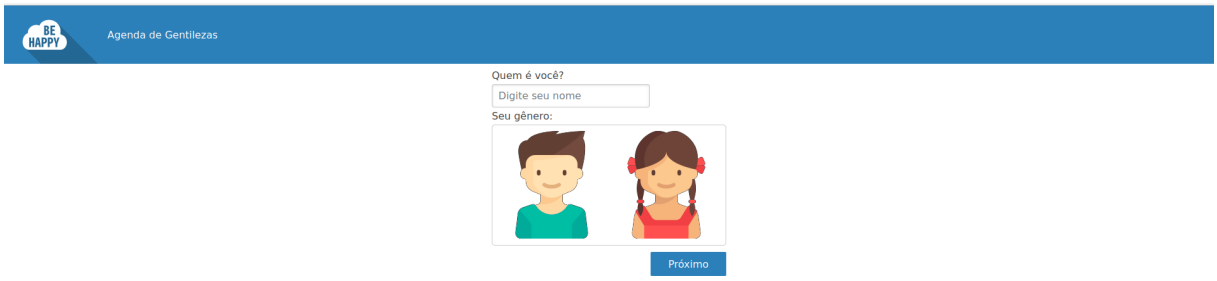


Figura 19 – Projeto desenvolvido com React - Cadastro de Usuário.

Fonte: Produzido pelo autor

Figura 20 – Projeto desenvolvido com React - Cadastro de Usuário.

Fonte: Produzido pelo autor

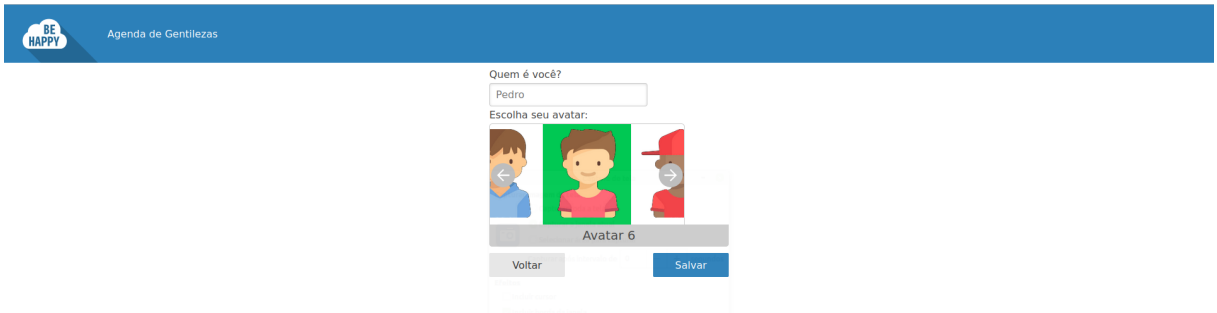


Figura 21 – Projeto desenvolvido com React - Cadastro de Usuário.

Fonte: Produzido pelo autor

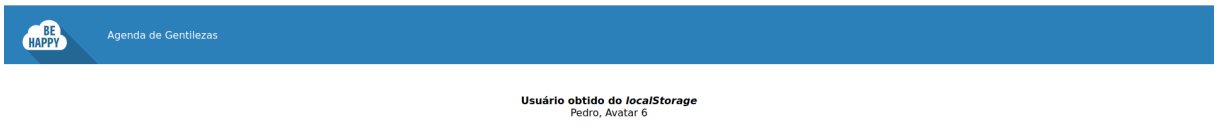


Figura 22 – Projeto desenvolvido com React - Cadastro de Usuário.

Fonte: Produzido pelo autor

3.1.2.1 VueJS

O projeto desenvolvido para o *framework* VueJS foi um gerenciador de notas, com login e cadastro, semelhantes ao que o sistema operacional *Windows* possui com sistema de "Notas Autoadesivas". As figuras 23, 24, 25, 26, 27, 28 e 29 mostram o projeto desenvolvido. Percebeu-se que, assim como o React e o Angular, o VueJs organiza seus "*controllers*" em *components* e também possui no mesmo arquivo do código-fonte uma parte de renderização da parte gráfica que são chamados de *template* e a parte da lógica que efetivamente controla o componente chamado de *script*. A Figura 30 mostra o código fonte do componente LVPainel. Observe que outros componentes podem ser concatenados dentro de um outro componente (no caso, os componentes LvNovaNota e LvListaNotas), característica presente também no React e a Angular.

Figura 23 – Projeto desenvolvido com VueJS - Tela de Login.

Fonte: Produzido pelo autor

Figura 24 – Projeto desenvolvido com VueJS - Cadastro de Usuário.

Fonte: Produzido pelo autor

Figura 25 – Projeto desenvolvido com VueJS - Cadastro de Usuário.

Fonte: Produzido pelo autor

Figura 26 – Projeto desenvolvido com VueJS - Tela de Login.

Fonte: Produzido pelo autor

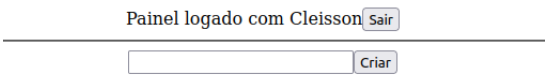


Figura 27 – Projeto desenvolvido com VueJS - Tela de Cadastro de Notas.

Fonte: Produzido pelo autor

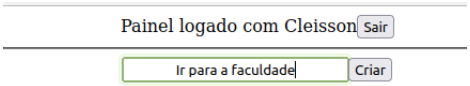


Figura 28 – Projeto desenvolvido com VueJS - Tela de Cadastro de Notas.

Fonte: Produzido pelo autor

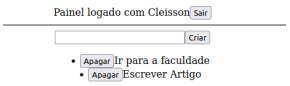


Figura 29 – Projeto desenvolvido com VueJS - Tela de Cadastro de Notas.

Fonte: Produzido pelo autor

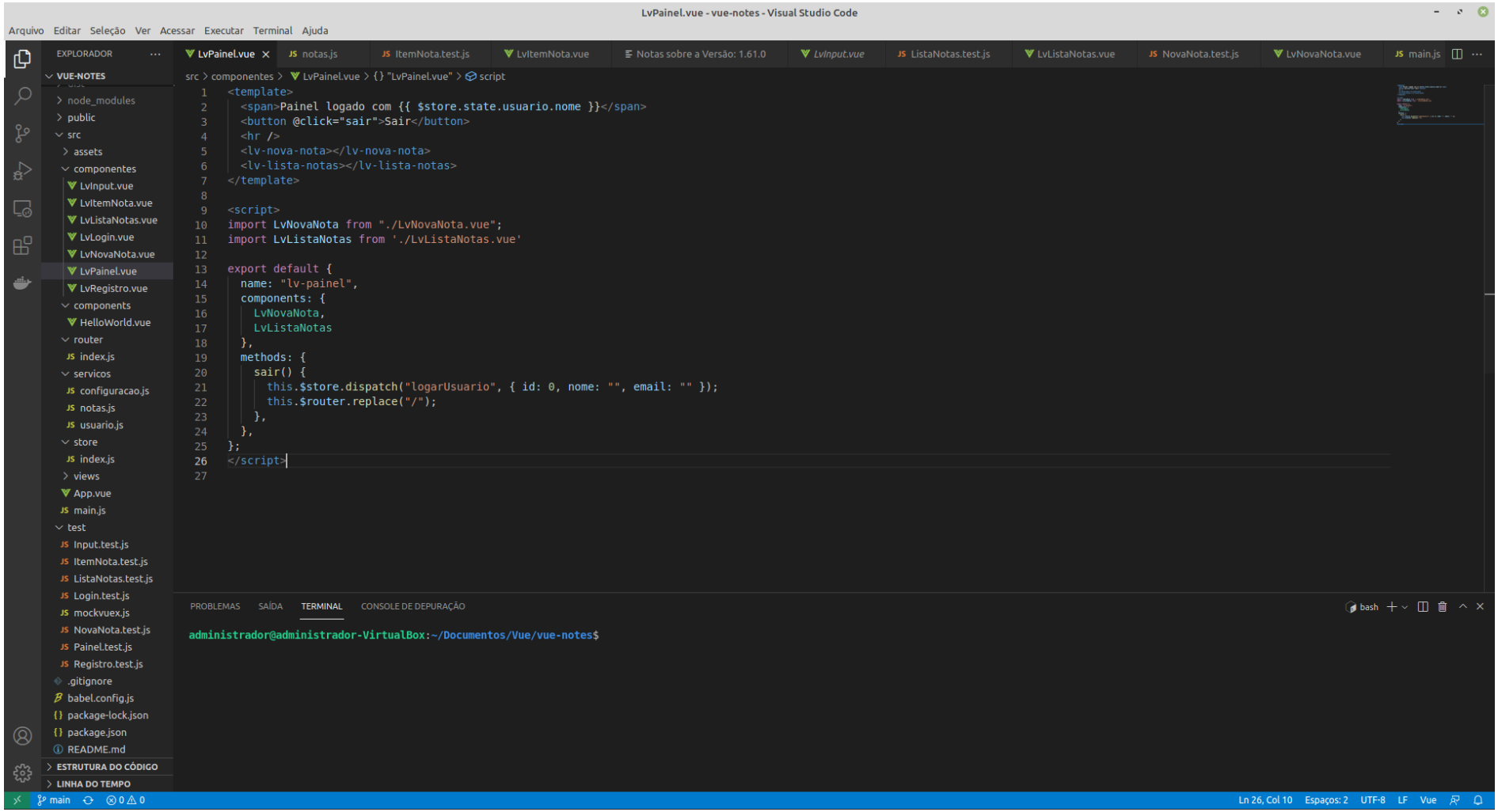


Figura 30 – Código fonte do projeto desenvolvido com VueJS demonstrando um *component*.

Fonte: Produzido pelo autor

3.2 Características comuns observadas e Esboços das Melhorias para o Método FrameWeb

A partir dos projetos e estudos dos *frameworks* descritos na Seção 3.1, foram percebidas características comuns que estão detalhadas a seguir.

3.2.1 Component

O *component* nos três *frameworks* SPA equivalem ao *controller* dos *frameworks* MVC tradicionais, mesclados com a camada de visualização. Eles são compostos por duas partes, uma parte lógica responsável por controlar as operações da *view* e a passagem de dados para ela, e uma parte gráfica que diz como o componente será construído e estruturado graficamente. Essas duas partes podem estar no mesmo arquivo ou em arquivos separados dependendo do *framework* e da escolha do programador. Além disso, os *components* são construídos de forma modular e podem ser reutilizados como parte de um outro *component*. As listagens 3.1, 3.2, 3.3 e 3.4 mostram um exemplo básico de *component* para cada *framework* estudado com suas respectivas partes gráficas e lógicas.

Listagem 3.1 – Exemplo de *component* em Angular, retirado do código-fonte do projeto desenvolvido.

```
1  import { Departamento } from '../../../models/departamento.model';
2  import { DepartamentoService } from '../../../services/departamento.service';
3  import { Component, OnInit } from '@angular/core';
4  import { Observable } from 'rxjs';
5  import { FormGroup, FormBuilder, Validators, FormControl } from '@angular/forms
       ';
6  import Swal from 'sweetalert2';
7
8  @Component({
9    selector: 'app-departamento',
10   templateUrl: './departamento.component.html',
11   styleUrls: ['./departamento.component.css']
12 })
13 export class DepartamentoComponent implements OnInit {
14
15   departamentos$: Observable<Departamento[]>;
16   edit: boolean;
17   displayDialogDepartamento: boolean;
18   form: FormGroup;
19
20   constructor(private departamentoService: DepartamentoService, private fb:
         FormBuilder) { }
21
22   ngOnInit() {
23     this.departamentos$ = this.departamentoService.list()
24     this.configForm()
25   }
26
27   configForm() {
```

```
28        this.form = this.fb.group({
29          id: new FormControl(),
30          nome: new FormControl('', Validators.required),
31          telefone: new FormControl('')
32        })
33      }
34
35      add() {
36        this.form.reset();
37        this.edit = false;
38        this.displayDialogDepartamento = true;
39      }
40
41      selecionaDepartamento(depto: Departamento) {
42        this.edit = true;
43        this.displayDialogDepartamento = true;
44        this.form.setValue(depto);
45      }
46
47      save() {
48        this.departamentoService.createOrUpdate(this.form.value)
49        .then(() => {
50          this.displayDialogDepartamento = false;
51          Swal.fire(`Departamento ${!this.edit ? 'salvo' : 'atualizado'} com
                sucesso.`, '', 'success')
52        })
53        .catch((erro) => {
54          this.displayDialogDepartamento = false;
55          Swal.fire(`Erro ao ${!this.edit ? 'salvar' : 'atualizar'} o departamento
                .`, `Detalhes: ${erro}`, 'error')
56        })
57        this.form.reset()
58      }
59
60      delete(depto: Departamento) {
61        Swal.fire({
62          title: 'Confirma a exclusão do departamento?',
63          text: "",
64          icon: 'warning',
65          showCancelButton: true,
66          confirmButtonText: 'Sim',
67          cancelButtonText: 'Não'
68        }).then((result) => {
69          if (result.value) {
70            this.departamentoService.delete(depto.id)
71            .then(() => {
72              Swal.fire('Departamento excluído com sucesso!', '', 'success')
73            })
74          }
75        })
76      }
77    }
```

Listagem 3.2 – Exemplo da parte gráfica do *component* descrito na Listagem 3.1.

```
1   <div class="card-header">
```

```
2   <h3> Departamentos
3   <button type="button" style="margin-right: 0px"
4   (click)="add()" class="text-right btn btn-outline-info btn-lg">
5   <i class="fa fa-plus-circle" aria-hidden="true"></i>
6   </button>
7   </h3>
8   </div>
9   <table class="table table-stripped table-hover table-borderd col-centred">
10  <thead class="thead-dark">
11  <tr>
12  <th class="text-center">Nome</th>
13  <th class="text-center">Telefone</th>
14  <th class="text-center">Ações</th>
15  </tr>
16  </thead>
17  <tbody>
18  <tr *ngFor="let departamento of departamentos$ | async">
19  <td class="text-center">{{departamento.nome}}</td>
20  <td class="text-center">{{departamento.telefone}}</td>
21  <td class="text-center">
22  <button type="button" (click)="delete(departamento)"
23  class="btn btn-danger">
24  <i class="fas fa-trash"></i>
25  </button>
26  </td>
27  </tr>
28  </tbody>
29  </table>
30  <a [routerLink]="['/admin/painel']" class="btn btn-primary">
31  <i class="fa fa-search" aria-hidden="true"></i>Voltar
32  </a>
33  <p-dialog header="Dados do departamento" [style]="{ width: '80vw' }" [
        contentStyle]="{'overflow': 'visible'}"
34  [(visible)]="displayDialogDepartamento" [responsive]="true" [modal]="true">
35  <div class="ui-grid ui-grid-responsive ui-fluid" *ngIf="form.value">
36  <form [formGroup]="form" class="p-fluid p-formgrid p-grid">
37  <div class="p-field p-col-12 p-md-6">
38  <label for="nome">Nome*:</label>
39  <input type="text" pInputText formControlName="nome" />
40  </div>
41  <div class="p-field p-col-12 p-md-6">
42  <label for="telefone">Telefone:</label>
43  <p-inputMask formControlName="telefone" mask="(99)9999-9999"></p-inputMask>
44  </div>
45  </form>
46  <button [disabled]="!form.valid" type="button" class="btn btn-primary" (click)=
        "save()">
47  <i class="fas fa-check-circle"></i> {{edit ? 'Atualizar' : 'Salvar'}}
48  </button>
49  </div>
50
51  </p-dialog>
```

Listagem 3.3 – Exemplo de *component* em React, retirado do código-fonte do projeto desenvolvido.

```
1  import React from 'react';
2  import Input from '../Input';
3  import Label from '../Label';
4  import GenderSelector from '../GenderSelector';
5  import Usuario from '../../models/Usuario';
6  import Button from '../Button';
7  import Toast from '../Toast'
8  import ImageScroller from '../ImageScroller'
9  import Avatar from '../../models/Avatar';
10
11
12 class NovoUsuario extends React.Component {
13   constructor(props) {
14     super(props);
15     this.state = {
16       usuario: new Usuario(),
17       validacao: {
18         nomeInvalido: false,
19         generoInvalido: false
20       },
21       primeiraVisaoCompleta: false
22     };
23   }
24
25   atualizarGenero(e, genero) {
26     e.preventDefault();
27     let usuario = this.state.usuario;
28     usuario.genero = genero;
29     this.setState({
30       usuario: usuario
31     });
32   }
33
34   atualizarNome(e) {
35     let usuario = this.state.usuario;
36     usuario.nome = e.target.value;
37     this.setState({
38       usuario: usuario
39     });
40   }
41
42   validar(e) {
43     e.preventDefault();
44     let usuario = this.state.usuario;
45     let validacao = this.state.validacao;
46     validacao.nomeInvalido = !usuario.validarNome();
47     validacao.generoInvalido = !usuario.validarGenero();
48     let mensagem = '';
49     let primeiraVisaoCompleta = false;
50     if (validacao.nomeInvalido && validacao.generoInvalido) {
51       mensagem = 'Os campos nome e gênero estão inválidos!'
52     } else if (validacao.nomeInvalido) {
53       mensagem = 'Seu nome está inválido!'
54     } else if (validacao.generoInvalido) {
55       mensagem = 'Selecione seu gênero!'
56     } else {
57       primeiraVisaoCompleta = true;
```

```
58        }
59        if (!primeiraVisaoCompleta) {
60          this.props.erro(mensagem);
61        }
62        this.setState({
63          validacao: validacao,
64          primeiraVisaoCompleta: primeiraVisaoCompleta
65        });
66      }
67
68      renderizarNome() {
69        return (
70        <section>
71        <Label
72        htmlFor="nome"
73        texto="Quem é você?"
74        valorInvalido={this.state.validacao.nomeInvalido}
75        />
76        <Input
77        id="nome"
78        placeholder="Digite seu nome"
79        maxLength="40"
80        readOnly={this.state.primeiraVisaoCompleta}
81        valorInvalido={this.state.validacao.nomeInvalido}
82        defaultValue={this.state.usuario.nome}
83        onChange={this.atualizarNome.bind(this)}
84        />
85        </section>
86        )
87      }
88
89      renderizarGenero() {
90        if (this.state.primeiraVisaoCompleta) {
91          return null
92        } else {
93          return (
94          <section>
95          <Label
96          texto="Seu gênero:"
97          valorInvalido={this.state.validacao.generoInvalido}
98          />
99          <GenderSelector
100         valorInvalido={this.state.validacao.generoInvalido}
101         genero={this.state.usuario.genero}
102         atualizarGenero={this.atualizarGenero.bind(this)}
103         />
104         </section>
105         )
106       }
107     }
108
109     renderizarBotoes() {
110       if (this.state.primeiraVisaoCompleta) {
111         return (
112         <section>
113         <Button
114         texto="Voltar"
```

```
115         onClick={e => {
116             e.preventDefault();
117             let usuario = this.state.usuario
118             usuario.avatar = Avatar.obterTodos()[0];
119             this.setState({
120               usuario: usuario,
121               primeiraVisaoCompleta: false
122             });
123         }}
124         />
125         <Button
126         principal
127         texto="Salvar"
128         onClick={e => {
129             e.preventDefault()
130             this.props.onSubmit(this.state.usuario)
131         }}
132         />
133         </section>
134         )
135       } else {
136         return (
137         <section>
138         <Button
139         principal
140         texto="Próximo"
141         onClick={this.validar.bind(this)}
142         />
143         </section>
144         )
145       }
146     }
147
148     atualizarGenero(e, genero) {
149       e.preventDefault();
150       let usuario = this.state.usuario;
151       usuario.genero = genero;
152       usuario.avatar = Avatar.obterTodos()[0];
153       this.setState({
154         usuario: usuario
155       });
156     }
157     renderizarAvatar() {
158       if (this.state.primeiraVisaoCompleta) {
159         return (
160         <section>
161         <Label
162         texto="Escolha seu avatar:"
163         />
164         <ImageScroller
165         arquivo="img/avatars.png"
166         eixoY={(this.state.usuario.genero == 'm' ? 0 : 1)}
167         elementos={Avatar.obterTodos()}
168         selecionado={this.state.usuario.avatar}
169         onChange={avatar => {
170             let usuario = this.state.usuario;
171             usuario.avatar = avatar;
```

```
172             this.setState({
173               usuario: usuario
174             });
175         }}
176         />
177         </section>
178         )
179       } else {
180         return null
181       }
182     }
183
184
185     render() {
186       return (
187       <div className="center">
188       <form className="pure-form pure-form-stacked">
189       {this.renderizarNome()}
190       {this.renderizarGenero()}
191       {this.renderizarAvatar()}
192       {this.renderizarBotoes()}
193       </form>
194       </div>
195       );
196     }
197   }
198
199   export default NovoUsuario;
```

Listagem 3.4 – Exemplo de *component* em VueJS, retirado do código-fonte do projeto desenvolvido.

```
1   <template>
2   <span>Painel logado com {{ $store.state.usuario.nome }}</span>
3   <button @click="sair">Sair</button>
4   <hr />
5   <lv-nova-nota></lv-nova-nota>
6   <lv-lista-notas></lv-lista-notas>
7   </template>
8
9   <script>
10  import LvNovaNota from "./LvNovaNota.vue";
11  import LvListaNotas from './LvListaNotas.vue'
12
13  export default {
14    name: "lv-painel",
15    components: {
16      LvNovaNota,
17      LvListaNotas
18    },
19    methods: {
20      sair() {
21        this.$store.dispatch("logarUsuario", { id: 0, nome: "", email: "" });
22        this.$router.replace("/");
23      },
24    },
```

```
25    };
26   </script>
```

3.2.2 Camada de serviços e de modelo

Os *frameworks* estudados não possuem de forma obrigatória uma camada de modelo e de serviço em sua especificação, ficando a cargo do desenvolvedor implementá-las se assim o desejar e da forma que desejar. Esses *frameworks* se "preocupam mais" com a parte do *front-end*, deixando a parte do *back-end* a cargo de outros *frameworks* (que podem inclusive ser implementados pelo próprio desenvolvedor caso não queira utilizar *frameworks* de terceiros).

Sendo assim, a principal diferença entre os *frameworks* estudados e os *frameworks* MVC é que nestes todas as camadas são definidas e organizadas pelo próprio *framework* de forma "obrigatória" enquanto que naqueles os *frameworks* se "preocupam" apenas com a parte *front-end*, mais especificamente com o *controller*, *view* e a navegação entre eles, tudo isso sendo feito por meio dos *components*. As demais partes do projeto ficam a cargo do desenvolvedor e a comunicação com o *back-end* é feita geralmente por meio do protocolo REST.

3.2.3 Esboços de propostas de alterações

Com vistas a contribuir com a linguagem de modelagem do método FrameWeb, foram construídos modelos como esboços do Modelo de Navegação do método utilizando a ferramenta *Visual Paradigm*.[3] Neles, foram já incorporados novos elementos que posteriormente viriam a ser propostos por esse trabalho, descritos em detalhes nas seções 3.3 e 3.4. As figuras 31, 32 e 33 mostram os modelos de navegação construídos.

[3] <https://www.visual-paradigm.com/>

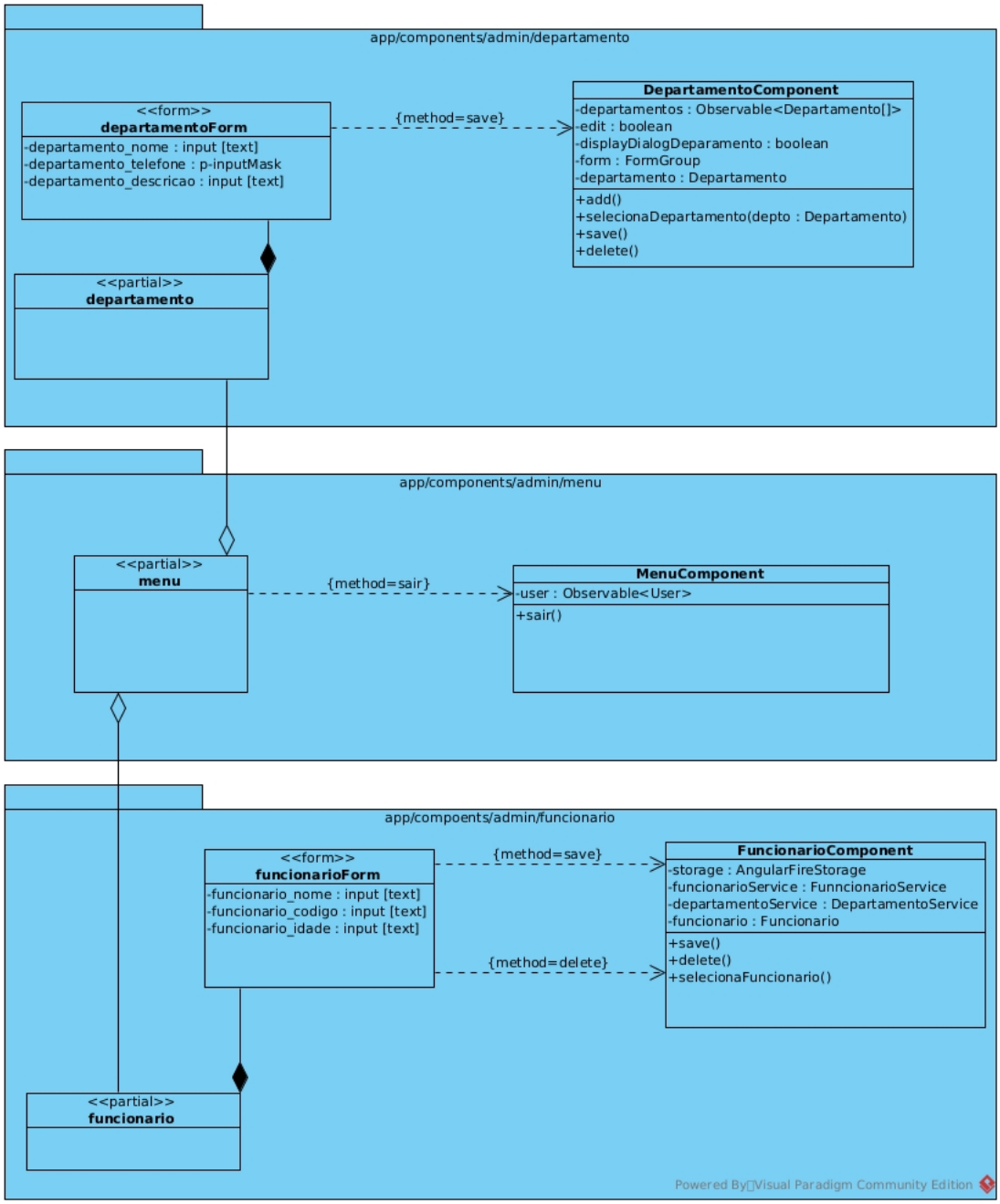


Figura 31 – Exemplo de modelo de navegação com a proposta incorporada

Fonte: Produzido pelo autor

A Figura 31 mostra uma parte do projeto de cadastro de requisições de departamentos de uma determinada empresa, feito com o *framework* Angular. Nele é possível ver o *partial* "menu" que contém links de acesso para o o *partial* "funcionário" e *partial* "departamento". Ambos os *partials* possuem um formulário de cadastro associados a eles e que são geridos pelas partes "*controller*" de seus *components* (a saber, o "FuncionarioComponent" e o "DepartamentoComponent" respectivamente).

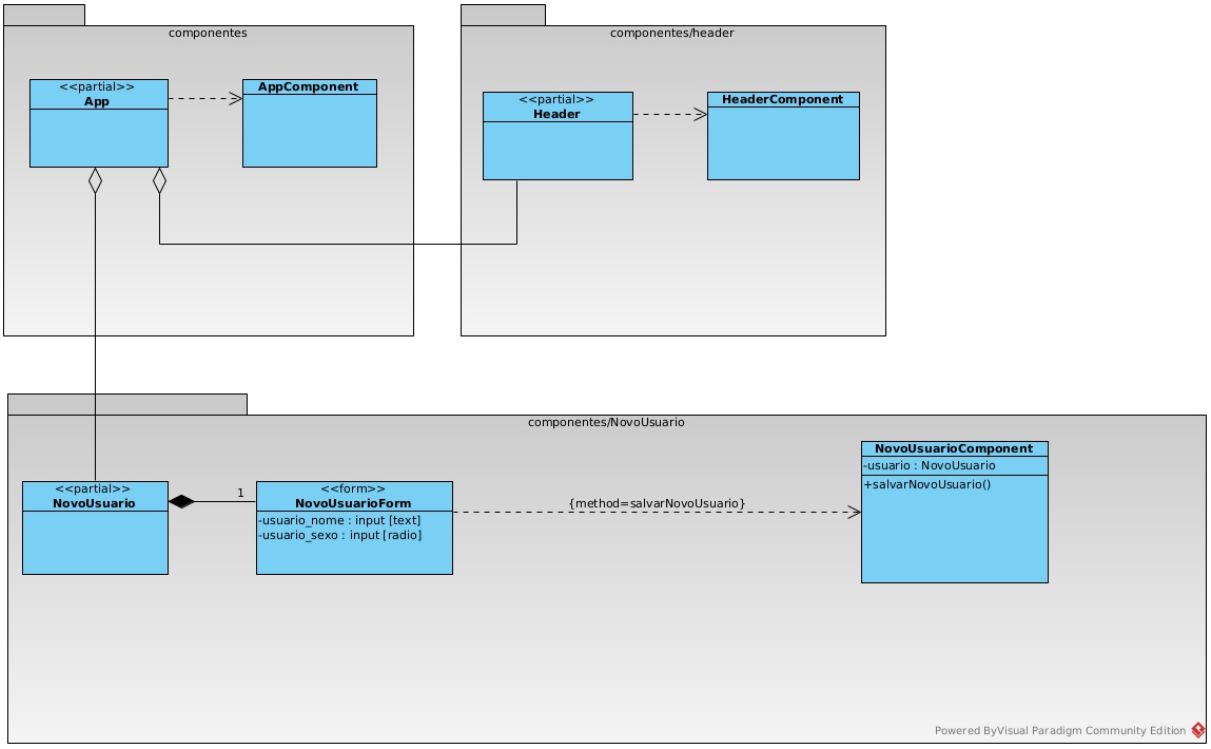


Figura 32 – Exemplo de modelo de navegação com a proposta incorporada

Fonte: Produzido pelo autor

A Figura 32 mostra uma parte do projeto da mini rede social, feito com o *framework* React, mais precisamente a parte de cadastro de um novo usuário. O *partial* "App" é "página principal" do aplicativo. Nele temos contido o *partial* "novo_usuario" que contém um formulário de cadastro de um novo usuário que é gerenciado pela parte "*controller*" de "novo_usuario", a saber "NovoUsuarioComponent". Observe também que "App" possui o *component* "Header" associado a ele, novamente evocando a questão da modularidade de *componets*.

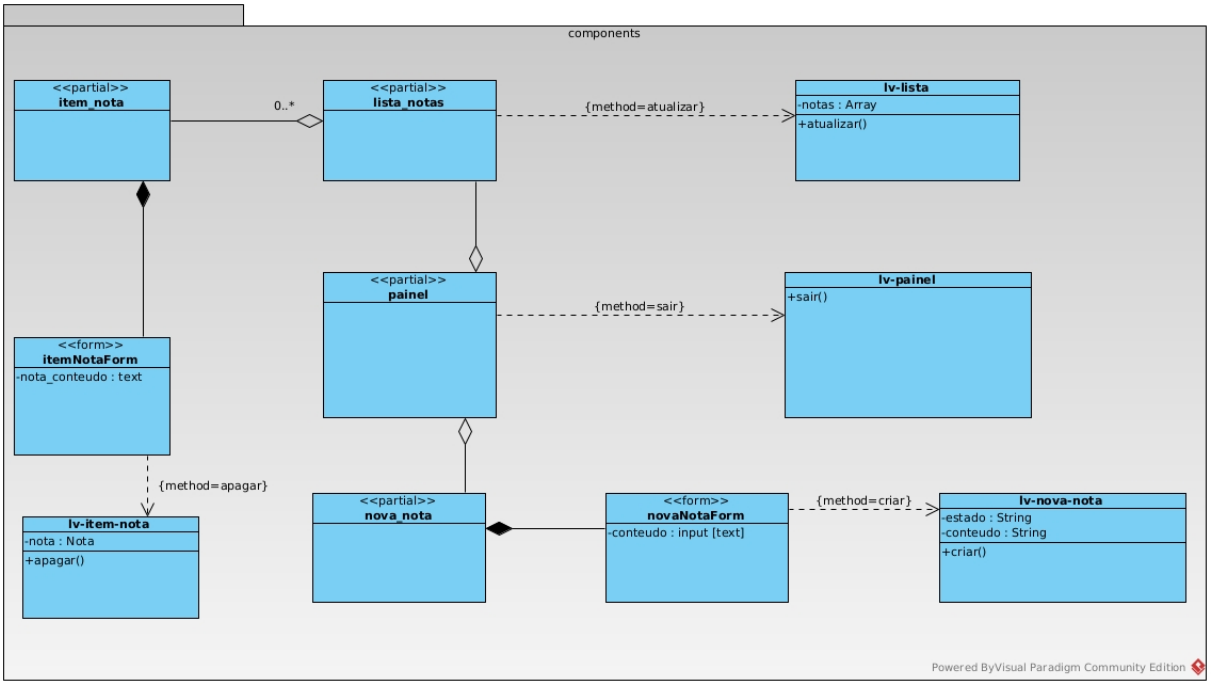


Figura 33 – Exemplo de modelo de navegação com a proposta incorporada

Fonte: Produzido pelo autor

A Figura 33 mostra uma parte do projeto de criação de notas semelhante ao sistema de notas autoadesivas do sistema "Windows", construído com o *framework* VueJS. Nele temos o *component* "painel" que representa a "página principal" do aplicativo. Atrelado a ele temos o o *component* "nova_nota" que é reponsável pela parte de cadastro de notas, e o *component* "item_notas" que é responsável por mostrar as notas cadastradas e atualizá-las. Junto ao "item_notas" temos atrelado o *component* "item_nota" que representa uma "nota" individualmente, sendo gerados no HTML a partir do número de notas cadastradas. No caso de não ter notas cadastradas, nada é exibido (ou gerado). Observe também que é a "parte *controller*" do *component* "item_nota" a responsável por remover uma nota cadastrada.

3.3 Análise: FrameWeb e os *Frameworks* SPA

Desde sua proposta original (SOUZA, 2007), o método FrameWeb propõe o uso de estereótipos da UML nas classes do Modelo de Navegação (conforme exemplificado na Figura 5) para representar diferentes elementos que compõem a arquitetura dos *frameworks* MVC tradicionais (Controladores Frontais), conforme descreve a Tabela 1. Associações entre estes diferentes elementos representam as interações entre eles, conforme descreve a Tabela 2.

Após o estudo dos *frameworks* Angular, React e Vue.js observou-se que a principal

Tabela 1 – Estereótipos UML que podem ser usados no Modelo de Navegação, segundo proposta original do FrameWeb (SOUZA, 2007).

Estereótipo	**O que representa**
(nenhum)	Uma classe de ação, para a qual o framework Front Controller delega a execução da ação.
«page»	Uma página Web estática ou dinâmica.
«template»	Um modelo (*template*) de uma página Web, processado por um *template engine* para produzir um documento HTML.
«form»	Um formulário HTML.
«binary»	Qualquer arquivo binário que pode ser exibido pelo navegador Internet (imagens, relatórios, documentos etc.).

Tabela 2 – Associações do Modelo de Navegação e o que representam, segundo proposta original do FrameWeb (SOUZA, 2007).

De	**Para**	**O que representa**
Página/modelo	Classe de ação	Um link entre a página/modelo e a classe de ação. Quando o link é seguido, a ação é executada.
Formulário	Classe de ação	Os dados do formulário são enviados à classe de ação para processamento.
Classe de ação	Página/modelo	A página ou modelo são exibidos como resultado da execução de uma determinada ação.
Classe de ação	Arquivo binário	Um arquivo binário é exibido como resultado da execução de uma determinada ação.
Classe de ação	Classe de ação	Uma outra classe de ação é executada como resultado da execução de uma primeira.

característica destes *frameworks* é a presença de um elemento denominado *component*,[4] que desempenha tanto a função de *controller* (controlador) quanto de *view* (visão). Tal elemento não é encontrado nos *frameworks* MVC tradicionais e, portanto, não foi incluído na proposta original de FrameWeb. Além disso, constatou-se que estes *frameworks* de fato se "preocupam" apenas com a parte do *front-end*, mais especificamente com o controlador, a visão e a interação entre eles, que é feita por meio dos *components*. As demais partes do projeto ficam a cargo do desenvolvedor e a comunicação com o *back-end* é realizada por meio do protocolo HTTP, comumente seguindo o modelo de arquitetura REST.

A Figura 34 mostra um exemplo de *component* implementado em Vue.js. Ele é parte de uma aplicação de criação de notas de texto (semelhante ao sistema de notas autoadesivas do Windows). A parte da visão é definida pela *tag* <template />, que define em seu conteúdo elementos HTML (e.g., e <button />) e referências a outros

[4] O termo *component* ou componente é usado em muitos contextos, com diferentes significados, na Engenharia de Software. Optamos, no entanto, por manter o termo utilizado na prática pelos usuários dos *frameworks* SPA, utilizando sempre sua grafia em inglês – *component* – para descrever, ao longo do texto, este novo elemento arquitetural trazido por esta nova categoria de *frameworks*.

```
<template>
  <span>Painel logado com {{ $store.state.usuario.nome }}</span>
  <button @click="sair">Sair</button>
  <hr />
  <lv-nova-nota></lv-nova-nota>
  <lv-lista-notas></lv-lista-notas>
</template>

<script>
import LvNovaNota from "./LvNovaNota.vue";
import LvListaNotas from './LvListaNotas.vue'

export default {
  name: "lv-painel",
  components: {
    LvNovaNota,
    LvListaNotas
  },
  methods: {
    sair() {
      this.$store.dispatch("logarUsuario", { id: 0, nome: "", email: "" });
      this.$router.replace("/");
    },
  },
};
</script>
```

Figura 34 – Exemplo de *component* do *framework* Vue.js.

components (e.g., <lv-nova-nota /> e <lv-lista-notas />). Já a parte controladora é definida pela *tag* <script />, que em seu conteúdo traz código JavaScript que define o nome do *component* (lv-painel), bem como os diferentes aspectos de controle, como outros *components* utilizados, métodos definidos neste *component*, etc. Nota-se, também, a interação entre visão e controle, por exemplo por meio da interpolação {{ $store.state.usuario.nome }}, que preenche o conteúdo da *tag* com o nome do usuário ou o atributo @click="sair" da *tag* <button />, que aciona o método sair() definido no controlador ao se clicar no botão contido na visão.

Para efetivarem consultas ou modificações nos dados do sistema de informação Web, os *components* realizam chamadas por meio do protocolo HTTP, comumente seguindo o modelo de arquitetura REST, a serviços oferecidos pelo *back-end*. Isso é feito pela parte controladora de um *component*, intermediado pelo *framework* SPA, a um serviço que encontra-se fora do escopo desta categoria de *frameworks* (inclusive podem ser utilizados outros *frameworks* para implementação do *back-end*, como o já citado *Express*, ou até mesmo uma plataforma e linguagem de programação diferentes da utilizada no *front-end*). No exemplo da Figura 34, o método sair() chama o serviço logarUsuario, do *back-end*, passando alguns parâmetros como parte de sua lógica de controle.

Os demais *frameworks* SPA possuem construtos e funcionamento similares, o que

Tabela 3 – Nova tabela de estereótipos para o Modelo de Navegação (atualiza a Tabela 1).

Estereótipo	**O que representa**
(nenhum)	Uma controladora de um *framework* Front Controller ou a parte controladora de um *component* de um *framework* SPA.
«page»	Uma página Web estática ou dinâmica.
«partial»	Parte de uma página HTML que é gerada em tempo de execução por meio de AJAX.
«form»	Um formulário HTML.
«binary»	Qualquer arquivo binário que pode ser exibido pelo navegador Internet (imagens, documentos, etc.).

nos levou à conclusão, a partir deste estudo, da necessidade de se propor alterações na linguagem de modelagem de FrameWeb para permitir o projeto de aplicações que utilizem este tipo de *framework*.

3.4 Propostas para o FrameWeb

Esta seção elenca as diversas propostas de alteração para o Modelo de Navegação de FrameWeb para inclusão de suporte aos *frameworks* SPA. As propostas serão ilustradas pelo exemplo da Figura 35, que modela a aplicação de anotação à qual o *component* exemplificado na Figura 34 faz parte.

Alteração 01: inclusão do novo estereótipo UML «partial» que pode ser usado em uma classe do modelo, para que esta represente um "pedaço" de página HTML que consiste da parte visão de um *component* de uma aplicação SPA. Classes com estereótipo «partial» (doravante denominadas *partials*) podem estar associadas a classes sem estereótipo (definidas a seguir), classes com estereótipo «form» (formulários HTML — funcionando da mesma forma que a associação entre páginas e formulários proposta originalmente) e outras classes «partial» (*partials* que se referem a *components* importados e reutilizados — vide Alteração 04).

Alteração 02: classes sem estereótipo, que anteriormente representavam classes controladoras (cf. Tabela 1[5]) dos *frameworks* MVC tradicionais (Controladores Frontais), agora representam, quando utilizados os *frameworks* SPA, a parte controladora de um *component*. Desta forma, a linguagem FrameWeb se mantém a mesma independente do tipo de *framework* Web utilizado. Importante observar que o artefato de *back-end* chamado pelas partes controladoras dos *components* são representados apenas no Modelo de Aplicação de FrameWeb, o que também já ocorre quando são usados *frameworks* MVC tradicionais. As alterações aqui propostas se limitam ao Modelo de Navegação.

[5] Originalmente as classes de controle eram denominadas "classes de ação" pelo FrameWeb. A partir de suas primeiras evoluções, o termo "classe de controle" ou "classe controladora" passou a ser utilizado e é o termo utilizado neste artigo.

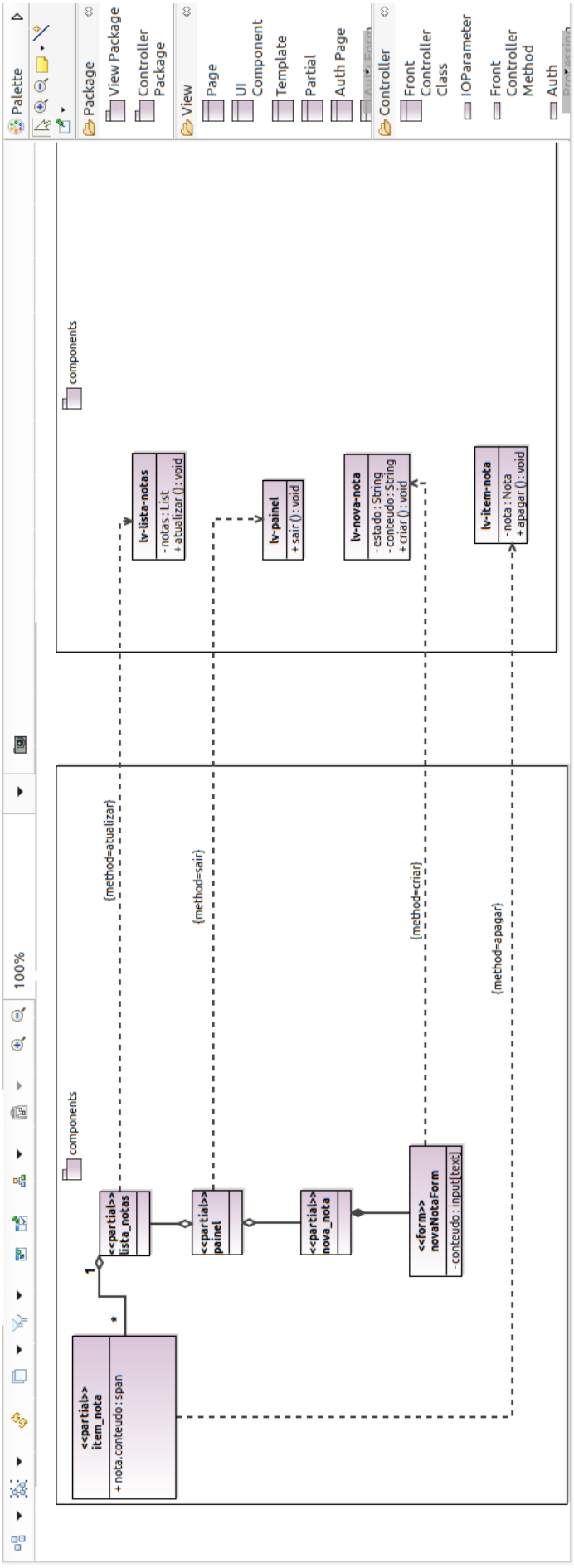


Figura 35 – Exemplo de modelo de navegação com suporte aos *frameworks* SPA já incorporado ao Editor FrameWeb.

Tabela 4 – Nova tabela de associações para o Modelo de Navegação (atualiza a Tabela 2).

De	**Para**	**O que representa**
Página ou *partial*	Controladora	Um link (ou componente visual equivalente) entre a página ou *partial* e a controladora (classe ou parte controladora de um *component*). Quando o link é utilizado, um método da controladora é chamado.
Formulário	Controladora	Botão (ou componente visual equivalente) que, quando utilizado, envia os dados do formulário à controladora (classe ou parte controladora de um *component*) e um método da controladora é chamado.
Controladora	Página ou *partial*	A página ou *partial* é exibida como resultado da chamada de um determinado método na controladora.
Controladora	Arquivo binário	Um arquivo binário é exibido como resultado da execução da chamada de um determinado método na controladora.

Uma importante observação sobre as alterações descritas acima é que um mesmo *component*, como por exemplo aquele ilustrado na Figura 34, é representado por dois elementos distintos no Modelo de Navegação: sua parte visão (*partial*) e sua parte controladora, associados entre si por meio de uma ou mais relações de dependência da UML, que representam chamadas de métodos (restrição method) do controlador por parte da visão. Na Figura 35, o *component* da Figura 34 é representado pelas classes painel e lv-painel (por convenção, o nome do *component* é usado como nome de sua parte controladora no diagrama) e a relação entre as duas representa a chamada do método sair(). Note que a representação da parte controladora de um *component* pode existir mesmo que não haja nenhuma chamada de método entre eles (neste caso, insere-se uma associação de dependência sem a definição da restrição method).

Alteração 03: originalmente, quando uma página chamava um controlador de um *framework* MVC tradicional, havia uma dependência no sentido do controlador representando a chamada e outra no sentido contrário representando o retorno, i.e., para que página o usuário seria direcionado. No entanto, no caso de SPAs o mais comum, na prática, é que os controladores dos *components*, ao processar um método, retornem para o mesmo *partial*. Para esses casos, sugere-se não ser necessário fazer a associação de retorno para o *partial* original, apenas sendo necessário caso o retorno do método redirecione para outro *partial*. Na Figura 35, todas as chamadas de método retornam ao mesmo *partial*.

Ainda neste contexto, a navegabilidade entre os *components* de uma aplicação SPA pode ser feita por redirecionamento dentro da própria *partial* ou como o retorno de um método da parte controladora do *component*.

Alteração 04: *components* trabalham também com a ideia de modularidade e um determinado *component* pode ser usado como parte de um *component* "maior" (ou mais complexo), que utiliza outros *components*, como "pecinhas de LEGO®" usadas para

construir estruturas maiores. Por exemplo, na Figura 34 o *component* lv-painel utiliza os *components* lv-nova-nota e lv-lista. Propõe-se, portanto, o uso da relação de agregação da UML entre esses elementos. Na Figura 35, observa-se as relações de agregação que painel (*partial* de lv-painel) tem com lista_notas (*partial* de lv-lista) e nova_nota (*partial* de lv-nova-nota).

Alteração 05: por fim, com o estudo observou-se que existem bibliotecas externas aos *frameworks* com *components* já prontos para uso (por exemplo, o PrimeNG para Angular, o PrimeReact para React e o PrimeVue para Vue.js, produzidos pela mesma organização que desenvolveu o PrimeFaces para JSF, citado na Seção 2.3). Para esses casos e para o caso em que o desenvolvedor/modelador desejar abstrair o funcionamento de um determinado *component* (ou seja, colocá-lo como "caixa preta") propõe-se que tais *components* sejam representados não como classes "importadas" por meio de associações de agregação mas sim como atributos do *partial* onde serão utilizados. Por exemplo, no formulário novaNotaForm do *partial* nova_nota, o campo de texto simples (input [text]) conteudo poderia ser substituído pelo *component* Editor do PrimeVue.[6]

3.5 Implementação

As propostas descritas anteriormente foram implementadas em FrameWeb por meio de alterações em seu metamodelo, que define sua linguagem de modelagem. Desta forma, serviram posteriormente de base para atualização das ferramentas FrameWeb (editor gráfico e gerador de código).

Como preparação para as alterações no metamodelo, foram atualizadas as tabelas 1 e 2 originais do FrameWeb. A Tabela 3 descreve o conjunto de estereótipos que podem agora ser utilizados no Modelo de Navegação, enquanto a Tabela 4 explica o que significam agora as associações entre os diferentes elementos deste modelo.

Além do suporte aos *frameworks* SPA, neste trabalho aproveitamos para atualizar estas definições relativas ao Modelo de Navegação nos seguintes sentidos: (1) foi retirado o elemento *template*, pois com o uso do método ao longo dos anos observou-se que o elemento página poderia representá-lo, sem necessidade de diferenciação no modelo; (2) o nome "Classe de Ação" usado pelo primeiro *framework* para o qual o FrameWeb proveu suporte foi substituído pelo termo "Controladora", podendo representar tanto uma classe controladora de um *framework* MVC tradicional quanto a parte controladora de um *component* de uma aplicação SPA; e (3) a inclusão dos *partials* contempla também os *frameworks* MVC tradicionais que possuam suporte a AJAX, já que os *partials* representariam trechos HTML da página principal (também HTML) geradas por requisição AJAX e os métodos operados sobre eles seriam feitos pelas classes controladoras tradicionais.

[6] <https://primefaces.org/primevue/editor>

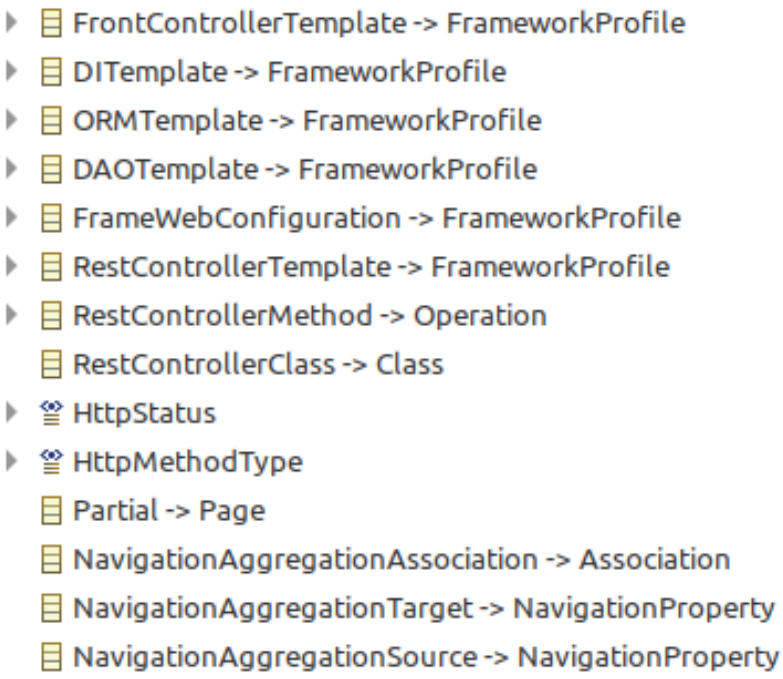


Figura 36 – Trecho do metamodelo de FrameWeb com a implementação das alterações propostas.

As modificações feitas no metamodelo do FrameWeb, que usa o Eclipse Modeling Framework (EMF), foram as seguintes:

- Para representar um *partial*, foi adicionada a metaclasse Partial como subclasse de Page, por se comportar exatamente como esta última, com a adição de suas particularidades no campo semântico (descritos anteriormente na seção 3.4);
- Para representar a associação de agregação entre *partials*, foi criada a metaclasse NavigationAggregationAssociation como subclasse de Association da UML2;
- NavigationAggregationAssociation é acompanhada pelos elementos NavigationAggregationSource e NavigationAggregationTarget, que representam respectivamente o todo e a parte da agregação. Estes herdam de NavigationProperty (do metamodelo original de FrameWeb), que herda de Property, da UML2, meta-classe que define o atributo de cardinalidade da relação utilizada para a associação de agregação.

Após realizadas as alterações no metamodelo do FrameWeb, foram também implementadas as devidas modificações para o Modelo de Navegação no editor de código, a saber: a adição do elemento Partial como uma classe do modelo e a possibilidade de adição de atributos dispostos dentro do Partial; adição do elemento que representa a relação de agregação de dois *partials* com o símbolo característico de losango não preenchido. A Figura 36 mostra as alterações feitas no metamodelo do método FrameWeb enquanto a Figura 37 mostra as adições feitas na ferramenta Sirius. O resultado dessa implementação

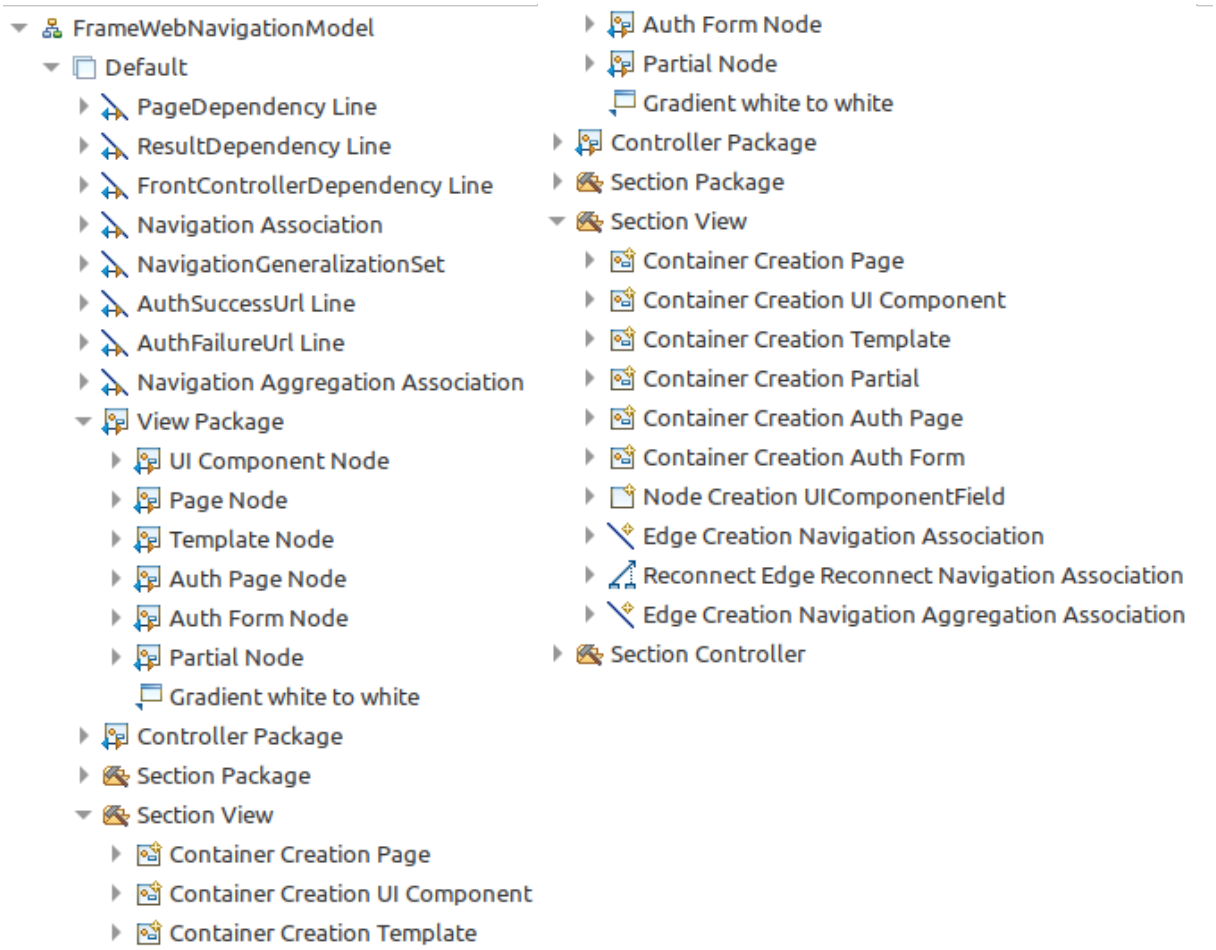


Figura 37 – Trecho da ferramenta Sirius com as mudanças no editor de código implementadas.

pode ser observado no Modelo de Navegação ilustrado na Figura 35, já apresentado anteriormente na Seção 3.4 para ilustrar as alterações propostas.

Por fim, foram implementadas também modificações no gerador de código, de modo que fosse possível realizar uma avaliação das propostas deste trabalho, descritas a seguir. Todas as modificações encontram-se disponíveis no repositório de código do FrameWeb.

4 Avaliação do Trabalho

Neste capítulo apresentamos os testes realizados para avaliar as propostas descritas no Capítulo 3. Na Seção 4.1 apresentamos a formalização do nosso experimento aplicando a abordagem GQM (*Goal Question Metric*) proposta por Basili, Caldiera e Rombach (1994). Na Seção 4.2 apresentamos a descrição completa dos experimentos e sua execução. Na Seção 4.3 apresentamos os resultados obtidos e finalmente na Seção 4.4 descrevemos as ameças à validade dos experimentos.

4.1 Formalização do Experimento

Para avaliar o suporte aos *frameworks* SPA, foi realizado um experimento em laboratório. Aplicando a abordagem GQM (*Goal Question Metric*) proposta por Basili, Caldiera e Rombach (1994), o objetivo é avaliar o suporte aos *frameworks* SPA dentro do contexto do método FrameWeb com os construtos propostos por este trabalho do ponto de vista de um modelador/desenvolvedor de *software*, como detalhado na Tabela 5.

Tabela 5 – Objetivo conforme o GQM.

Objetivo	**Propósito:**	Avaliar
	Questão:	o suporte aos *frameworks* SPA
	Objeto:	dentro do contexto do método FrameWeb com os construtos propostos por este trabalho
	Ponto de Vista:	sob o ponto de vista de um modelador/desenvolvedor de *software*

Fonte: Produzido pelo autor

Baseado neste objetivo, surgem três questões para serem respondidas: (1) se uma aplicação SPA simples pode ser devidamente modelada utilizando FrameWeb e suas ferramentas; (2) se estas ferramentas conseguem gerar código esqueleto para a aplicação modelada; e (3) quanto código, em média, consegue ser gerado automaticamente a partir do modelo, liberando o desenvolvedor de fazê-lo manualmente. Para responder essas questões, utilizamos a métrica de número de *tags* geradas pelo método após a modelagem conforme descrito na Tabela 6.

4.2 Execução

Para conduzir a avaliação, foram utilizados o método FrameWeb e suas ferramentas, agora modificadas para incluir a nova categoria de *frameworks* (conforme proposto por

Tabela 6 – Questões e métrica conforme o GQM.

Questão:	Uma aplicação SPA simples pode ser devidamente modelada utilizando FrameWeb e suas ferramentas?
Questão:	Estas ferramentas conseguem gerar código esqueleto para a aplicação modelada?
Questão:	Quanto código, em média, consegue ser gerado automaticamente a partir do modelo?
Métrica:	Número de *Tags* geradas pelo gerador de código após a modelagem com os novos recursos do método FrameWeb

Fonte: Produzido pelo autor

este trabalho), para modelar 7 sistemas de informação Web (WIS) implementados por desenvolvedores diferentes e que utilizaram um dos três *frameworks* SPA mencionados na Seção 3.1. Os WISs foram implementados nos últimos anos como trabalhos da disciplina sobre Programação Web chamada DWWS (Desenvolvimento Web e Web Semântica), oferecida anualmente a estudantes de graduação e pós-graduação em nossa universidade. O critério de escolha foram projetos desenvolvidos a partir de 2020. Até esta data, foram desenvolvidos 7 projetos no contexto de *frameworks* SPA. Feito o modelo FrameWeb, foi executada a geração de código, foi comparado o total de *tags* HTML geradas com o número de *tags* codificadas manualmente nas implementações feitas pelos estudantes e foi calculada a razão (número de *tags* geradas divido por número de *tags* codificadas manualmente) em porcentagem da parte *view* dos *components*. Isso foi feito pois o gerador de código do método FrameWeb, por definição, gera o código esqueleto das aplicações modeladas pelo método e os elementos da camada de visualização especificados no modelo de navegação. Por isso só foi comparado o número de *tags* gerados já que o gerador de código gerará somente as assinaturas dos métodos e a definição dos atributos, e não sua implementação que caberá ser preenchida pela pessoa desenvolvedora.

Para cada *framework* foram implementados *templates* que serão disponibilizados na versão final do método FrameWeb com as implementações do suporte SPA proposto neste trabalho. A Tabela 7 mostra a relação dos projetos testados, *framework* utilizado, seu link original (projeto implementado pelos alunos) e o link com as implementações, modelagens e geração de código do FrameWeb SPA para os testes.

Os projetos podem ser considerados de média complexidade, com muitos *components* do tipo CRUD (telas de cadastro, com funções *Create, Retrieve, Update, Delete*) mas também com algumas exibições gráficas mais complexas como, por exemplo, animação de botões.

A modelagem com o FrameWeb SPA para a avalição foi feita pelo autor baseando-se no código-fonte dos *components* de cada projeto. Como as melhorias deste trabalho se concentram no Modelo de Navegação, foram modelados apenas os Modelos de Navegação e

Tabela 7 – Projetos avaliados.

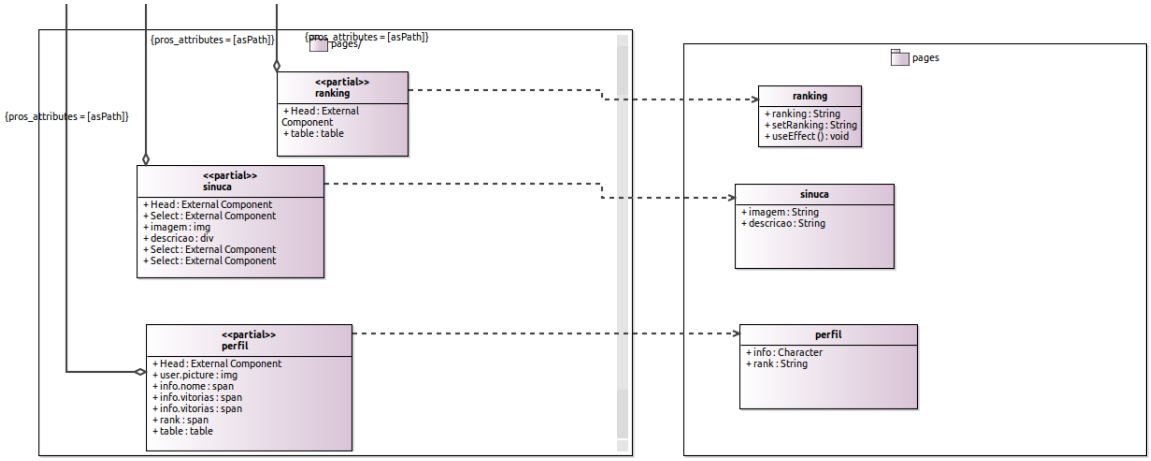
Projeto	**Framework**	**Link Original**	**Link Avaliação**
Virtual Pool	React	<https://github.com/dwws-ufes/2021-VirtualPool>	<https://github.com/pedrohbh/2021-VirtualPool>
Ufes Sports	Angular	<https://github.com/dwws-ufes/2021-UFESports>	<https://github.com/pedrohbh/2021-UFESports>
FeiranaMao	Angular	<https://github.com/dwws-ufes/2020-feiranamao>	<https://github.com/pedrohbh/2020-feiranamao>
TasteUfes	VueJS	<https://github.com/dwws-ufes/2020-TasteUfes>	<https://github.com/pedrohbh/2020-TasteUfes>
UMDB	React	<https://github.com/dwws-ufes/2020-UMDb>	<https://github.com/pedrohbh/2020-UMDb>
UfesPay	React	<https://github.com/dwws-ufes/2020-UfesPay>	<https://github.com/pedrohbh/2020-UfesPay>
ClassiWeb	React	<https://github.com/dwws-ufes/2020-ClassiWeb>	<https://github.com/pedrohbh/2020-ClassiWeb>

Fonte: Produzido pelo autor

o Modelo de Entidades para poder usar os elementos de entidade no Modelo de Navegação.

A Figura 38 mostra um exemplo com um trecho da modelagem do projeto "Virtual Pool", nele podemos ver os *components* "sinuca", "ranking" e "perfil" dentro da pasta "pages". Podemos ver tanto a parte *view* (representado pelos *partials*) como a parte *controller* (representado pelo *Front Controller*) e a ligação entre eles. Observe também as relações de agregação (losango aberto) que representam o uso de outros *components* dentro de *component* e o "pros_attributes", que representa os atributos do *component* filho (no exemplo, o nome do atributo se chama "asPath") que está sendo usado dentro do *component* pai no qual se passará algum dado do pai para o filho.

Figura 38 – Trecho da Modelagem do projeto "Virtual Pool"



Fonte: Produzido pelo autor

A seguir, são apresentados os resultados do experimento.

4.3 Resultados

Com um total de 91 *components* modelados e gerados pelo gerador de código, obtivemos uma média de 69% e mediana de 66,7% do total de *tags* gerados pelo gerador de código. Com relação à parte *controller* do *component*, o gerador de código gera apenas as assinaturas dos métodos e dos atributos, cabendo ao programador desenvolver sua lógica.

As tabelas 8 e 9 apresentam os resultados dos experimentos divididos por projeto e por *component*. Nas tabelas temos a coluna `Projeto` que indica a qual projeto determinado *component* (descrito na coluna `Arquivo`) pertence. A coluna `T. Ori.` corresponde ao número de *tags* originalmente codificadas pelos autores originais dos projetos enquanto que a coluna `T. Ger.` corresponde ao número de *tags* geradas pelo gerador de código do método FrameWeb. A coluna `Porcentagem` corresponde a razão de *tags* geradas por *tags* originais em porcento.

Um fator importante com relação ao experimento é que como o gerador de código, dependendo do tipo de *tag* modelada, pode gerar uma *tag* simples ou *tag* dupla (por exemplo, <input type="text" /> seria uma *tag* simples enquanto que ... seria uma *tag* dupla), e da mesma forma o desenvolvedor original pode utilizar uma *tag* simples ou uma *tag* dupla conforme sua conveniência, para o nosso teste, contamos as *tags* apenas com o número de *tags* de abertura (no exemplo descrito anteriormente, foi contado somente o <input type="text" /> e o , não sendo contado portanto o). Isso é representado pelas colunas "Tags Original" (T. Ori.) e "Tags Gerados" (T. Ger.) da Tabela 9 representando respectivamente as *tags* de abertura desenvolvidas manualmente pelos autores dos projetos e as *tags* de abertura geradas pelo gerador de código. Já as mesmas colunas das *tags* da Tabela 8 representam as contagens totais de todas as *tags* (tanto de abertura quanto de fechamento de ambos os projetos). A coluna porcentagem contêm a relação "Tags Gerado" / "Tags Original" para as duas tabelas. Para efeito de análise, foi considerada a porcentagem da versão "formatada" (Tabela 9) conforme justificado anteriormente. As últimas linhas das tabelas contêm a média por coluna e a mediana por coluna.

A Tabela 11 no Apêndice A mostra a tabela completa dos testes com todos os *components* de todos os projetos modelados com método FrameWeb já com as melhorias propostas por este trabalho.

Tabela 8 – Resultado da Avaliação com as *Tags* Originais

Projeto	**Arquivo**	**T. Ori.**	**T. Ger.**	**Porgentagem**
Virtual Pool	Header.js	46	24	52.17%
Virtual Pool	sinuca.js	25	15	60.0%
Virtual Pool	jogo.js	21	10	47.62%

Tabela 8 – Resultado da Avaliação com as *Tags* Originais

Projeto	**Arquivo**	**T. Ori.**	**T. Ger.**	**Porgentagem**
Virtual Pool	rdf.js	21	9	42.86%
Virtual Pool	perfil.js	67	25	37.31%
Virtual Pool	entrar.js	25	7	28.0%
Virtual Pool	index.js	20	7	35.0%
Virtual Pool	ranking.js	31	15	48.39%
Virtual Pool	jogar.js	30	17	56.67%
Ufes Sports	footer.component.html	19	6	31.58%
Ufes Sports	header.component.html	18	8	44.44%
Ufes Sports	showevent.component.html	32	18	56.25%
Ufes Sports	login.component.html	24	15	62.5%
Ufes Sports	signup.component.html	40	15	37.5%
Ufes Sports	config.component.html	4	2	50.0%
Ufes Sports	student.component.html	32	22	68.75%
Ufes Sports	sport.component.html	39	23	58.97%
Ufes Sports	createevent.component.html	49	20	40.82%
Ufes Sports	myevent.component.html	26	14	53.85%
Ufes Sports	home.component.html	18	8	44.44%
FeiranaMao	admin-produto.component.html	126	45	35.71%
FeiranaMao	login.component.html	48	7	14.58%
FeiranaMao	success.component.html	9	2	22.22%
FeiranaMao	cart.component.html	54	29	53.7%
FeiranaMao	cart-item.component.html	12	8	66.67%
FeiranaMao	lojas.component.html	15	8	53.33%
FeiranaMao	produto.component.html	24	12	50.0%
FeiranaMao	dialog-produto-info.component.html	18	10	55.56%
FeiranaMao	navbar.component.html	33	2	6.06%
FeiranaMao	admin-loja.component.html	36	12	33.33%
FeiranaMao	usuario.component.html	92	35	38.04%
FeiranaMao	default.component.html	10	4	40.0%
TasteUfes	Overlay.vue	8	6	75.0%
TasteUfes	Footbar.vue	8	8	100.0%
TasteUfes	Login.vue	39	29	74.36%
TasteUfes	MenuMobile.vue	63	45	71.43%
TasteUfes	MenuOption.vue	24	24	100.0%

Tabela 8 – Resultado da Avaliação com as *Tags* Originais

Projeto	**Arquivo**	**T. Ori.**	**T. Ger.**	**Porgentagem**
TasteUfes	HelpButton.vue	16	16	100.0%
TasteUfes	UserCard.vue	54	38	70.37%
TasteUfes	Toolbar.vue	32	21	65.62%
TasteUfes	Snackbar.vue	8	8	100.0%
TasteUfes	EditButton.vue	6	6	100.0%
TasteUfes	DetailsButton.vue	6	6	100.0%
TasteUfes	DeleteButton.vue	28	26	92.86%
TasteUfes	NutritionFactsTable.vue	58	52	89.66%
TasteUfes	RecipeData.vue	56	50	89.29%
UMDB	Footer.js	6	4	66.67%
UMDB	MovieSuggestion.js	16	9	56.25%
UMDB	App.js	28	49	175.0%
UMDB	SingleReview.js	11	7	63.64%
UMDB	MovieInfo.js	43	17	39.53%
UMDB	AdminInternalHeader.js	8	5	62.5%
UMDB	MovieCard.js	14	16	114.29%
UMDB	AdminContainer.js	9	9	100.0%
UMDB	Header.js	12	6	50.0%
UMDB	DefaultForm.js	10	8	80.0%
UMDB	AdminSidebar.js	10	10	100.0%
UMDB	Filter.js	33	31	93.94%
UMDB	HomeContainer.js	14	14	100.0%
UfesPay	profile.jsx	46	23	50.0%
UfesPay	login.jsx	16	8	50.0%
UfesPay	createacc.jsx	24	10	41.67%
UfesPay	transferhistory.js	11	5	45.45%
UfesPay	navtop.jsx	21	16	76.19%
UfesPay	news.jsx	5	3	60.0%
UfesPay	menuitem.js	8	8	100.0%
UfesPay	layout.jsx	5	5	100.0%
UfesPay	navitem.js	4	4	100.0%
UfesPay	transferencia.js	6	2	33.33%
UfesPay	menu.js	8	5	62.5%
UfesPay	transfercard.jsx	49	43	87.76%
UfesPay	landingpage.jsx	4	4	100.0%
UfesPay	transfer.jsx	24	21	87.5%

Tabela 8 – Resultado da Avaliação com as *Tags* Originais

Projeto	**Arquivo**	**T. Ori.**	**T. Ger.**	**Porgentagem**
UfesPay	home.jsx	8	4	50.0%
ClassiWeb	StyledButton.tsx	3	2	66.67%
ClassiWeb	Footer.tsx	2	2	100.0%
ClassiWeb	AdCard.tsx	38	39	102.63%
ClassiWeb	MySelect.tsx	8	8	100.0%
ClassiWeb	Categories.tsx	1	2	200.0%
ClassiWeb	PageBase.tsx	8	7	87.5%
ClassiWeb	Panel.tsx	31	27	87.1%
ClassiWeb	Ads.tsx	44	35	79.55%
ClassiWeb	Address.tsx	8	8	100.0%
ClassiWeb	Carousel.tsx	8	10	125.0%
ClassiWeb	ProductState.tsx	1	2	200.0%
ClassiWeb	AppBar.tsx	43	28	65.12%
ClassiWeb	CategoriesList.tsx	17	7	41.18%
ClassiWeb	AdminPanel.tsx	11	10	90.91%
ClassiWeb	CategoriesList.tsx	7	7	100.0%
ClassiWeb	Ad.tsx	104	90	86.54%
ClassiWeb	Feedback.tsx	8	4	50.0%
Média	**Média**	**24.89**	**15.64**	**71.25%**
Mediana	**Mediana**	**18**	**10**	**65.12%**

Tabela 9 – Resultado da Avaliação com as *Tags* Formatadas

Projeto	**Arquivo**	**T. Ori.**	**T. Ger.**	**Porcentagem**
Virtual Pool	Header.js	26	12	46.15%
Virtual Pool	sinuca.js	15	8	53.33%
Virtual Pool	jogo.js	11	5	45.45%
Virtual Pool	rdf.js	11	5	45.45%
Virtual Pool	perfil.js	35	13	37.14%
Virtual Pool	entrar.js	13	4	30.77%
Virtual Pool	index.js	11	4	36.36%
Virtual Pool	ranking.js	16	8	50.0%
Virtual Pool	jogar.js	18	9	50.0%
Ufes Sports	footer.component.html	11	3	27.27%
Ufes Sports	header.component.html	9	4	44.44%
Ufes Sports	showevent.component.html	16	9	56.25%

Tabela 9 – Resultado da Avaliação com as *Tags* Formatadas

Projeto	**Arquivo**	**T. Ori.**	**T. Ger.**	**Porcentagem**
Ufes Sports	login.component.html	13	12	92.31%
Ufes Sports	signup.component.html	23	12	52.17%
Ufes Sports	config.component.html	2	2	100.0%
Ufes Sports	student.component.html	16	11	68.75%
Ufes Sports	sport.component.html	20	13	65.0%
Ufes Sports	createevent.component.html	27	15	55.56%
Ufes Sports	myevent.component.html	13	7	53.85%
Ufes Sports	home.component.html	9	4	44.44%
FeiranaMao	admin-produto.component.html	66	30	45.45%
FeiranaMao	login.component.html	27	5	18.52%
FeiranaMao	success.component.html	5	1	20.0%
FeiranaMao	cart.component.html	27	15	55.56%
FeiranaMao	cart-item.component.html	6	4	66.67%
FeiranaMao	lojas.component.html	8	4	50.0%
FeiranaMao	produto.component.html	12	6	50.0%
FeiranaMao	dialog-produto-info.component.html	9	5	55.56%
FeiranaMao	navbar.component.html	17	1	5.88%
FeiranaMao	admin-loja.component.html	20	9	45.0%
FeiranaMao	usuario.component.html	48	21	43.75%
FeiranaMao	default.component.html	5	2	40.0%
TasteUfes	Overlay.vue	4	3	75.0%
TasteUfes	Footbar.vue	4	4	100.0%
TasteUfes	Login.vue	21	19	90.48%
TasteUfes	MenuMobile.vue	32	23	71.88%
TasteUfes	MenuOption.vue	12	12	100.0%
TasteUfes	HelpButton.vue	8	8	100.0%
TasteUfes	UserCard.vue	27	19	70.37%
TasteUfes	Toolbar.vue	18	12	66.67%
TasteUfes	Snackbar.vue	4	4	100.0%
TasteUfes	EditButton.vue	3	3	100.0%
TasteUfes	DetailsButton.vue	3	3	100.0%
TasteUfes	DeleteButton.vue	14	13	92.86%
TasteUfes	NutritionFactsTable.vue	31	26	83.87%
TasteUfes	RecipeData.vue	30	25	83.33%

Tabela 9 – Resultado da Avaliação com as *Tags* Formatadas

Projeto	**Arquivo**	**T. Ori.**	**T. Ger.**	**Porcentagem**
UMDB	Footer.js	3	2	66.67%
UMDB	MovieSuggestion.js	9	5	55.56%
UMDB	App.js	25	25	100.0%
UMDB	SingleReview.js	6	4	66.67%
UMDB	MovieInfo.js	22	9	40.91%
UMDB	AdminInternalHeader.js	4	3	75.0%
UMDB	MovieCard.js	8	8	100.0%
UMDB	AdminContainer.js	5	5	100.0%
UMDB	Header.js	6	3	50.0%
UMDB	DefaultForm.js	6	5	83.33%
UMDB	AdminSidebar.js	5	5	100.0%
UMDB	Filter.js	20	16	80.0%
UMDB	HomeContainer.js	8	8	100.0%
UfesPay	profile.jsx	26	14	53.85%
UfesPay	login.jsx	9	5	55.56%
UfesPay	createacc.jsx	14	7	50.0%
UfesPay	transferhistory.js	6	3	50.0%
UfesPay	navtop.jsx	12	8	66.67%
UfesPay	news.jsx	3	2	66.67%
UfesPay	menuitem.js	4	4	100.0%
UfesPay	layout.jsx	3	3	100.0%
UfesPay	navitem.js	2	2	100.0%
UfesPay	transferencia.js	3	1	33.33%
UfesPay	menu.js	4	4	100.0%
UfesPay	transfercard.jsx	27	23	85.19%
UfesPay	landingpage.jsx	3	3	100.0%
UfesPay	transfer.jsx	15	14	93.33%
UfesPay	home.jsx	5	3	60.0%
ClassiWeb	StyledButton.tsx	2	1	50.0%
ClassiWeb	Footer.tsx	1	1	100.0%
ClassiWeb	AdCard.tsx	25	23	92.0%
ClassiWeb	MySelect.tsx	4	4	100.0%
ClassiWeb	Categories.tsx	1	1	100.0%
ClassiWeb	PageBase.tsx	5	4	80.0%
ClassiWeb	Panel.tsx	19	14	73.68%
ClassiWeb	Ads.tsx	25	18	72.0%

Tabela 9 – Resultado da Avaliação com as *Tags* Formatadas

Projeto	**Arquivo**	**T. Ori.**	**T. Ger.**	**Porcentagem**
ClassiWeb	Address.tsx	5	5	100.0%
ClassiWeb	Carousel.tsx	5	5	100.0%
ClassiWeb	ProductState.tsx	1	1	100.0%
ClassiWeb	AppBar.tsx	25	14	56.0%
ClassiWeb	CategoriesList.tsx	11	4	36.36%
ClassiWeb	AdminPanel.tsx	6	5	83.33%
ClassiWeb	CategoriesList.tsx	4	4	100.0%
ClassiWeb	Ad.tsx	58	47	81.03%
ClassiWeb	Feedback.tsx	5	2	40.0%
Média	**Média**	**13.64**	**8.62**	**69.04%**
Mediana	**Mediana**	**11**	**5**	**66.67%**

Os arquivos com os projetos, modelos e resultados estão disponíveis em <https://nemo.inf.ufes.br/projetos/frameweb/>. O Apêndice A contém a relação completa dos *components* separados por projeto com o total da porcentagem de *tags* gerados por *component* e também os originalmente codificados.

Consideramos os resultados destes experimentos satisfatórios, pois foi possível demonstrar que o método e as ferramentas FrameWeb podem ser utilizados para modelar aplicações SPA simples e gerar uma quantidade razoável de código, considerando que toda a parte de lógica da aplicação precisa necessariamente ser implementada manualmente por um desenvolvedor.

4.4 Ameaças à Validade e Limitações da Avaliação

Em relação a ameaças à validade, fizemos esta análise à luz do arcabouço proposto por Wohlin et al. (2012), a saber:

- *Conclusion Validity*: a avaliação foi feita com base em sete projetos de desenvolvimento de aplicações SPA, todos desenvolvidos em contexto acadêmico (*low statistical power*), o experimento foi conduzido pelo próprio autor da proposta, em busca de um resultado positivo (*fishing*), e os resultados dependem da modelagem dos sistemas na linguagem FrameWeb, que pode ser diferente se repetida por outra pessoa (*reliability of treatment implementation*). Para mitigar tais ameaças, todo o material (projetos, modelagem, resultados) foi disponibilizado de forma pública para que possam ser avaliados pelo leitor interessado;

- *Construct Validity*: apenas o número de *tags* geradas a partir dos modelos FrameWeb foi medido (*mono-method bias*), e o experimento foi feito pelo próprio autor da proposta, sendo sujeito da própria avaliação que conduzia (*interaction of testing and treatment*). Para mitigar esta última ameaça, utilizamos projetos de *software* desenvolvidos por outras pessoas como base para a modelagem FrameWeb;

- *External Validity*: os projetos de *software* usados como base foram todos desenvolvidos no contexto de uma disciplina universitária, portanto com características diferentes daqueles desenvolvidos na indústria (*Interaction of setting and treatment*). No entanto, de acordo com Höst, Regnell e Wohlin (2000), a diferença entre experimentos com estudantes e profissionais é pequena.

Destacamos com relação a limitações de nossa avaliação o não uso de testes com desenvolvedores/modeladores terceiros e consequentemente a aplicação de um questionário para poder avaliar questões de usabilidade e também ter uma avaliação mais robusta que demonstre a representatividade desta categoria de *framework* com as mudanças propostas por este trabalho. Além disso, também destacamos a limitação de não aplicar os métodos propostos por trabalhos relacionados (Seção 2.5) sobre os mesmos projetos desenvolvidos descritos anteriormente para podermos comparar a eficiência daqueles métodos com o nosso. Tais avaliações não foram possíveis no escopo deste trabalho e são sugeridos como trabalhos futuros para esta pesquisa.

5 Considerações Finais

O MDD demonstra ser uma técnica promissora para desenvolvimento de *software*, pois facilita o processo de documentação, projeto, planejamento e codificação, diminuindo a distância entre eles (ATKINSON; KUHNE, 2003; PASTOR; MOLINA, 2007; SELIC, 2003; BRAMBILLA; CABOT; WIMMER, 2017). Torna-se mais prático desenvolver um projeto desta forma do que fazer toda a documentação necessária do planejamento para depois fazer a codificação, bem como manter a documentação atualizada conforme o projeto progride.

O método FrameWeb tem se demonstrado ao longo dos anos uma ferramenta interessante e prática com grandes potenciais para uso industrial no futuro, unindo as práticas do MDD ao nicho do desenvolvimento dos WIS. Neste contexto, seu objetivo é providenciar suporte a todo tipo de *framework* utilizado para o desenvolvimento de sistemas de informação para a Web (SOUZA, 2020). Os *frameworks* SPA tem ganhado cada vez mais popularidade e, conforme demonstrado por esse trabalho, sua estrutura de organização e filosofia de código diferem dos *frameworks* MVC (no qual a construção do método FrameWeb foi baseado) o que motivou as adições aqui propostas.

Neste capítulo, demonstramos as contribuições e objetivos alcançados (Seção 5.1) bem como discutimos as limitações do trabalho e possíveis trabalhos futuros (Seção 5.2).

5.1 Objetivos Alcançados

Neste trabalho foram apresentadas as melhorias propostas ao método FrameWeb de forma que este passe a englobar a categoria de *frameworks* SPA. Foram apresentadas as propostas de alteração na linguagem FrameWeb, as modificações implementadas em seu metamodelo, bem como as alterações realizadas no editor gráfico e, por fim, no gerador de código, permitindo-nos avaliar preliminarmente a factibilidade e a utilidade das propostas, cumprindo-se assim o objetivo principal (proporcionar suporte aos *frameworks* SPA no método FrameWeb), bem como os objetivos específicos, a saber:

a) Levantar os elementos arquiteturais relevantes desta categoria de *frameworks* por meio dos estudos dos *frameworks* com o desenvolvimento de miniprojetos e estudo da documentação oficial de cada um dos mesmos;

b) Analisar o nível de suporte que o método FrameWeb já proporciona a alguns destes elementos por meio de uma análise comparativa entre a forma arquitetural dos *frameworks* SPA e a implementação do método FrameWeb até o momento antes do proposto dos novos construtos;

c) Propor uma evolução do método (novos construtos e ferramentas) de modo que possa dar suporte a *frameworks* SPA a saber os elementos descritos no Capítulo 3 como por exemplo a adição do elemento *Partial* de modo que tal categoria de *framework* possa ser devidamente modelada e utilizada por desenvolvedores/modeladores para projetos WIS que utilizam tal categoria de *framework*.

A Tabela 10 sumariza os objetivos específicos e os resultados.

Tabela 10 – Objetivos Específicos e Resultados Alcançados.

Objetivos Específicos	**Resultados**
Levantar os elementos arquiteturais desta categoria de *frameworks*	Estudos dos *frameworks* com o desenvolvimento de miniprojetos e estudo da documentação oficial de cada um dos mesmos (descritos no Capítulo 3)
Analisar o nível de suporte que o método FrameWeb já proporciona a alguns destes elementos	Estudo do método FrameWeb e comparação do suporte já provido pelo método com a dinâmica e forma de desenvolvimento dos *frameworks* SPA, este baseado em suas documentações oficiais e desenvolvimento dos mini projetos (Capítulos 2 e 3)
Propor uma evolução do método (novos construtos e ferramentas) de modo que possa dar suporte a *frameworks* SPA	Baseado nos estudos da etapa anterior, adição de suporte ao desenvolvimento com *frameworks* SPA pelo método FrameWeb com criação de novos construtos como, por exemplo, o elemento *Partial*, implementação da interface gráfico do Editor FrameWeb, geração de código e realização de testes para avaliar essa nova adição de suporte (Capítulos 3 e 4)

Fonte: Produzido pelo autor

Acreditamos que este trabalho contribui para a ampliação do uso do método FrameWeb e, consequentemente, para o uso do Desenvolvimento Orientado a Modelos (MDD) no contexto da Engenharia Web. O suporte aos *frameworks* SPA auxilia tanto os desenvolvedores como os projetistas na construção e manutenção de WIS que utilizem tais *frameworks*, bem como na comunicação entre as equipes, o que é preconizado pelo MDD (PASTOR; MOLINA, 2007; BRAMBILLA; CABOT; WIMMER, 2017).

5.2 Limitações e Trabalhos Futuros

Com relação às principais limitações do trabalho, apontamos as seguintes:

a) Para ser usado na prática, o atual gerador de código precisa de *templates* com a sintaxe específica de cada *framework* e, até o momento, foram escritos *templates* para os *frameworks* Angular, React e VueJS apenas;

b) O trabalho foi avaliado em laboratório (i.e., pelos próprios autores), utilizando projetos de sistemas de informação Web desenvolvidos por estudantes em

contexto acadêmico (trabalho prático em disciplina de Desenvolvimento Web). Devido a problemas logísticos, não conduzimos uma avaliação com profissionais na indústria (estudos de caso) ou outros tipos de avaliação mais robustas, portanto os resultados de nossa avaliação são preliminares em relação a alguns aspectos da proposta;

c) A proposta é baseada em 3 dos *frameworks* SPA mais utilizados, porém para dizer com mais segurança que há suporte à categoria dos *frameworks* SPA em FrameWeb, um estudo mais extenso como, por exemplo, a construção de uma ontologia desta categoria de *frameworks*, seria necessário;

d) A usabilidade das ferramentas (editor e gerador de código) FrameWeb ainda é muito baixa para que o método, incluindo as propostas deste trabalho, possam ser levadas à prática na indústria. São consideradas ainda protótipos, tornando difícil organizar testes em ambiente industrial de produção (vide item (b) acima).

Como sugestão de trabalhos futuros, portanto, temo: (i) aprimorar o gerador de códigos de modo a incluir mais *frameworks* SPA; (ii) experimentar o método em contextos reais de produção, avaliando-o com profissionais em desenvolvimento de sistemas de informação Web; (iii) construir uma ontologia dos *frameworks* SPA de modo a estabelecer com mais precisão os elementos arquiteturais relevantes desta categoria de *frameworks* e garantir que FrameWeb proveja um suporte completo a eles; (iv) melhorar a usabilidade das ferramentas FrameWeb para facilitar sua adoção por desenvolvedores. Todos essas sugestões de trabalhos futuros, com exceção do item (ii) já estão sendo implementados por meio de trabalhos de mestrado e iniciação científica. Todos eles visam a um aprimoramento da ferramenta e uma melhor base conceitual para os frameworks SPA e do emprego do método de MDD dentro do contexto do desenvolvimento de WIS.

Referências

AGüERO, J. et al. Mdd-approach for developing pervasive systems based on service-oriented multi-agent systems. *Advances in distributed computing and artificial intelligence journal*, Ediciones Universidad de Salamanca, Salamanca, v. 2, n. 3, p. 55–64, 2013. ISSN 2255-2863. Citado na página 32.

ALMEIDA, J. P. A. *Model-driven design of distributed applications*. Tese (Doutorado) — University of Twente, Netherlands, jun 2006. CTIT Ph.D.-Thesis Series, ISSN 1381-3617, No. 06-85 ; Telematica Instituut Fundamental Research Series, ISSN 1388-1795, No. 018. Citado 2 vezes nas páginas 19 e 20.

ALMEIDA, N. V. de; CAMPOS, S. L.; SOUZA, V. E. S. A Model-Driven Approach for Code Generation forWeb-based Information Systems Built with Frameworks. In: *Proc. of the 23rd Brazilian Symposium on Multimedia and the Web (WebMedia 2017)*. Gramado, RS, Brazil: ACM, 2017. p. 245–252. Citado 3 vezes nas páginas 14, 23 e 24.

ALUR, D.; CRUPI, J.; MALKS, D. *Core J2EE Patterns: Best Practices and Design Strategies*. 2nd. ed. [S.l.]: Prentice Hall / Sun Microsystems Press, 2003. Citado na página 23.

AMBLER, S. W. Agile model driven development is good enough. *IEEE Software*, IEEE, v. 20, n. 5, p. 71–73, 2003. Citado na página 20.

ATKINSON, C.; KUHNE, T. Model-driven development: a metamodeling foundation. *IEEE Software*, v. 20, n. 5, p. 36–41, 2003. Citado 5 vezes nas páginas 8, 19, 20, 21 e 78.

BARCELLOS, M. et al. Organizing empirical studies as learning iterations in design science research projects. In: *Proceedings of the XXI Brazilian Symposium on Software Quality*. New York, NY, USA: Association for Computing Machinery, 2023. (SBQS '22). ISBN 9781450399999. Disponível em: <https://doi.org/10.1145/3571473.3571474>. Nenhuma citação no texto.

BASILI, V. R.; CALDIERA, G.; ROMBACH, D. H. The goal question metric approach. *Encyclopedia of software engineering*, p. 528–532, 1994. Citado na página 67.

BETTINI, L. *Implementing Domain Specific Languages with Xtext and Xtend - Second Edition*. 2nd. ed. [S.l.]: Packt Publishing, 2016. ISBN 1786464969. Citado na página 22.

BEZERRA, J. D. H.; SOUZA, C. T. de. A model-based approach to generate reactive and customizable user interfaces for the web of things. In: *Proceedings of the 25th Brazillian Symposium on Multimedia and the Web*. New York, NY, USA: Association for Computing Machinery, 2019. (WebMedia '19), p. 57–60. ISBN 9781450367639. Disponível em: <https://doi.org/10.1145/3323503.3360631>. Citado na página 31.

BONIFÁCIO, R. et al. Neoidl: A domain-specific language for specifying rest services. In: *SEKE*. [S.l.: s.n.], 2015. p. 613–618. Citado 2 vezes nas páginas 30 e 31.

BRAMBILLA, M.; CABOT, J.; WIMMER, M. Mdse principles. In: _____. *Model-Driven Software Engineering in Practice*. Cham: Springer International Publishing, 2017. ISBN

978-3-031-02549-5. Disponível em: <https://doi.org/10.1007/978-3-031-02549-5>. Citado 2 vezes nas páginas 78 e 79.

CAMPOS, S. L.; SOUZA, V. E. S. FrameWeb Editor: Uma Ferramenta CASE para suporte ao Método FrameWeb. In: *Anais do 16º Workshop de Ferramentas e Aplicações, 23º Simpósio Brasileiro de Sistemas Multimedia e Web (WFA/WebMedia 2017)*. Gramado, RS, Brazil: SBC, 2017. p. 199–203. Citado 3 vezes nas páginas 14, 23 e 24.

COSTA-SORIA, C. et al. Model-Driven Development of Aspect-Oriented Software Architectures. *Journal of Universal Computer Science*, Verlag der Technischen Universität Graz, v. 19, n. 10, p. 1433–1473, 2013. ISSN 0948-695X. Citado na página 32.

DELJOUYI, A.; RAMSIN, R. Mdd4rest: Model-driven methodology for developing restful web services. In: *MODELSWARD*. [S.l.: s.n.], 2022. p. 93–104. Citado na página 31.

DEURSEN, A. V.; KLINT, P. Little languages: little maintenance? *Journal of Software Maintenance: Research and Practice*, Wiley Online Library, v. 10, n. 2, p. 75–92, 1998. Citado na página 20.

ED-DOUIBI, H. et al. Emf-rest: generation of restful apis from models. In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. [S.l.: s.n.], 2016. p. 1446–1453. Citado 2 vezes nas páginas 30 e 31.

FRAKES, W. B.; KANG, K. Software reuse research: Status and future. *IEEE Transactions on Software Engineering*, IEEE, v. 31, n. 7, p. 529–536, 2005. Citado na página 22.

HAUPT, F. et al. A model-driven approach for rest compliant services. In: *2014 IEEE International Conference on Web Services*. [S.l.: s.n.], 2014. p. 129–136. Citado 2 vezes nas páginas 30 e 31.

HERNANDEZ-MENDEZ, A.; SCHOLZ, N.; MATTHES, F. A model-driven approach for generating restful web services in single-page applications. In: *MODELSWARD*. [S.l.: s.n.], 2018. p. 480–487. Citado na página 30.

HEVNER, A. A three cycle view of design science research. *Scandinavian Journal of Information Systems*, v. 19, 01 2007. Citado na página 15.

HöST, M.; REGNELL, B.; WOHLIN, C. Using students as subjects – a comparative study of students and professionals in lead-time impact assessment. *Empirical Software Engineering*, n. 5, p. 201–214, 2000. Citado na página 77.

KIFOUCHE, A. et al. Model driven framework to enhance sensor network design cycle. *European transactions on telecommunications*, Wiley Subscription Services, Inc, v. 30, n. 8, p. e3560–n/a, 2019. ISSN 2161-3915. Citado na página 32.

KLEPPE, A. G. et al. *MDA explained: the model driven architecture: practice and promise*. [S.l.]: Addison-Wesley Professional, 2003. Citado na página 20.

MACHADO, K. K. *Angular 11 e Firebase*. São Paulo, SP, Brasil: Casa do Código, 2019. Citado na página 35.

MARTINS, B. F. *Evolução do Método FrameWeb para o Projeto de Sistemas de*

Informação Web Utilizando uma Abordagem Dirigida a Modelos. Vitória, ES, Brasil, 2016. Citado 6 vezes nas páginas 8, 19, 20, 21, 22 e 24.

MARTINS, B. F.; SOUZA, V. E. S. A Model-Driven Approach for the Design of Web Information Systems based on Frameworks. In: *Proc. of the 21st Brazilian Symposium on Multimedia and the Web (WebMedia 2015)*. Manaus, AM, Brazil: ACM, 2015. p. 41–48. Disponível em: <http://dl.acm.org/citation.cfm?id=2820439>. Citado 5 vezes nas páginas 8, 14, 20, 22 e 25.

MATTOS, D. P.; MUCHALUAT-SAADE, D. C. Multisem: A mulsemedia model for supporting the development of authoring tools. In: *Proceedings of the 24th Brazilian Symposium on Multimedia and the Web*. New York, NY, USA: Association for Computing Machinery, 2018. (WebMedia '18), p. 109–116. ISBN 9781450358675. Disponível em: <https://doi.org/10.1145/3243082.3243114>. Citado na página 31.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, ACM New York, NY, USA, v. 37, n. 4, p. 316–344, 2005. Citado na página 20.

MORADI, H.; ZAMANI, B.; ZAMANIFAR, K. CaaSSET: A Framework for Model-Driven Development of Context as a Service. *Future Generation Computer Systems*, Elsevier B.V, v. 105, p. 61–95, 2020. ISSN 0167-739X. Citado na página 32.

OLIVEIRA, Y.; SILVEIRA, L.; SOUZA, C. A model-driven approach to evolve recommender systems. In: *Proceedings of the 24th Brazilian Symposium on Multimedia and the Web*. New York, NY, USA: Association for Computing Machinery, 2018. (WebMedia '18), p. 169–172. ISBN 9781450358675. Disponível em: <https://doi.org/10.1145/3243082.3267457>. Citado na página 31.

PANDO, B.; CASTILLO, J. Plantumlgen: A tool for teaching model driven development. In: IEEE. *2022 17th Iberian Conference on Information Systems and Technologies (CISTI)*. [S.l.], 2022. p. 1–6. Citado na página 31.

PASPALLIS, N. An mdd-based method for building context-aware applications with high reusability. *Journal of software : evolution and process*, Wiley Subscription Services, Inc, Chichester, v. 31, n. 11, p. n/a, 2019. ISSN 2047-7473. Citado na página 32.

PASTOR, O.; MOLINA, J. C. *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. ISBN 978-3-540-71867-3. Disponível em: <http://link.springer.com/10.1007/978-3-540-71868-0>. Citado 3 vezes nas páginas 19, 78 e 79.

PINTO, T. D.; GONÇALVES, W. I.; COSTA, P. V. User interface prototype generation from agile requirements specifications written in concordia. In: *Proceedings of the 25th Brazillian Symposium on Multimedia and the Web*. New York, NY, USA: Association for Computing Machinery, 2019. (WebMedia '19), p. 61–64. ISBN 9781450367639. Disponível em: <https://doi.org/10.1145/3323503.3360639>. Citado na página 31.

PONCIANO, T. et al. A generative approach for android sensor-based applications. In: *Proceedings of the Brazilian Symposium on Multimedia and the Web*. New York, NY, USA: Association for Computing Machinery, 2020. (WebMedia '20), p. 33–40. ISBN

9781450381963. Disponível em: <https://doi.org/10.1145/3428658.3430976>. Citado na página 31.

PONTES, G. *Progressive Web Apps: Construa aplicações progressivas com React*. São Paulo, SP, Brasil: Casa do Código, 2018. Citado na página 35.

PRADO, R. C. do; SOUZA, V. E. S. Securing FrameWeb: Supporting Role-based Access Control in a Framework-based Design Method for Web Engineering. In: *Proc. of the 24th Brazilian Symposium on Multimedia and the Web (WebMedia '18)*. Salvador, BA, Brazil: ACM, 2018. p. 213–220. ISBN 9781450358675. Citado na página 31.

ROMANO, B. L.; CUNHA, A. M. d. A framework for web applications using an agile and collaborative model driven development (ac-mdd). *Acta scientiarum. Technology*, Editora da Universidade Estadual de Maringá - EDUEM, Maringa, v. 41, n. 1, p. 38349–e38349, 2019. ISSN 1806-2563. Citado na página 31.

ROSSI, D. Uml-based model-driven rest api development. In: *WEBIST (1)*. [S.l.: s.n.], 2016. p. 194–201. Citado na página 31.

SCOTT, E. *SPA Design and Architecture: Understanding Single Page Web Applications*. 1st. ed. USA: Manning Publications Co., 2015. ISBN 1617292435. Citado 5 vezes nas páginas 8, 26, 27, 28 e 29.

SELIC, B. The pragmatics of model-driven development. *IEEE software*, IEEE, v. 20, n. 5, p. 19–25, 2003. Citado 2 vezes nas páginas 19 e 78.

SOSA-REYNA, C. M.; TELLO-LEAL, E.; LARA-ALABAZARES, D. Methodology for the model-driven development of service oriented iot applications. *Journal of systems architecture*, Elsevier B.V, v. 90, p. 15–22, 2018. ISSN 1383-7621. Citado na página 32.

SOUSA, M. F. de et al. A generative software development approach for mulsemedia application domain. In: *Proceedings of the 25th Brazillian Symposium on Multimedia and the Web*. New York, NY, USA: Association for Computing Machinery, 2019. (WebMedia '19), p. 29–36. ISBN 9781450367639. Disponível em: <https://doi.org/10.1145/3323503.3360290>. Citado na página 31.

SOUZA, V. E. S. *FrameWeb: um Método baseado em Frameworks para o Projeto de Sistemas de Informação Web*. Dissertação (Mestrado) — Universidade Federal do Espírito Santo, 2007. Disponível em: <http://portais.ufes.br/PRPPG/ext/mono.php?progpess=2032&curso=9&prog=30001013007P0>. Citado 3 vezes nas páginas 10, 58 e 59.

SOUZA, V. E. S. The FrameWeb Approach to Web Engineering: Past, Present and Future. In: ALMEIDA, J. P. A.; GUIZZARDI, G. (Ed.). *Engineering Ontologies and Ontologies for Engineering*. 1. ed. Vitória, ES, Brazil: NEMO, 2020. cap. 8, p. 100–124. ISBN 9781393963035. Disponível em: <http://purl.org/nemo/celebratingfalbo>. Citado 7 vezes nas páginas 8, 14, 22, 24, 25, 34 e 78.

STOREY, M.-A. et al. Using a visual abstract as a lens for communicating and promoting design science research in software engineering. In: *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. [S.l.: s.n.], 2017. p. 181–186. Citado na página 15.

THOMAS, D. Mda: Revenge of the modelers or uml utopia? *IEEE software*, IEEE, v. 21,

n. 3, p. 15–17, 2004. Citado na página 20.

TRASK, B.; ROMAN, A. Using domain specific modeling in developing software defined radio components and applications. In: *ECOOP Workshop on Domain-Specific Program Development (DSPD), Nantes, France*. [S.l.: s.n.], 2006. Citado na página 20.

van GIGCH, J. P.; PIPINO, L. L. In search for a paradigm for the discipline of information systems. *Future Computing Systems*, v. 1, n. 1, p. 71–97, 1986. Nenhuma citação no texto.

VILARINHO, L. *Front-end com Vue.js: Da teoria à prática sem complicações*. São Paulo, SP, Brasil: Casa do Código, 2017. Citado na página 35.

WIERINGA, R. *Design science methodology for information systems and software engineering*. Netherlands: Springer, 2014. 10.1007/978-3-662-43839-8. ISBN 978-3-662-43838-1. Citado na página 15.

WOHLIN, C. et al. *Experimentation in software engineering*. [S.l.]: Springer Science & Business Media, 2012. Citado na página 76.

Apêndices

APÊNDICE A – Resultado dos Testes de Validação

A Tabela 11 mostra a tabela completa dos testes com todos os *components* de todos os projetos modelados com método FrameWeb já com as melhorias propostas por este trabalho. Conforme dito no Capítulo 4 e repetido aqui é que um fator importante com relação ao experimento é que como o gerador de código, dependendo do tipo de *tag* modelada, pode gerar uma *tag* simples ou *tag* dupla (por exemplo, <input type="text" /> seria uma *tag* simples enquanto que ... seria uma *tag* dupla), e da mesma forma o desenvolvedor original pode utilizar uma *tag* simples ou uma *tag* dupla conforme sua conveniência, para o nosso teste, contamos as *tags* apenas com o número de *tags* de abertura (no exemplo descrito anteriormente, foi contado somente o <input type="text" /> e o , não sendo contado portanto o). Isso é representado pelas colunas "TagsOriginalFormatado" e "TagsGeradoFormatado" representando respectivamente as *tags* de abertura desenvolvidas manualmente pelos autores dos projetos e as *tags* de abertura geradas pelo gerador de código. Já as colunas "TagsOriginal" e "TagsGerado" representam as contagens totais de todas as *tags* (tanto de abertura quanto de fechamento de ambos os projetos). As porcentagens contêm a relação "tagsGerado"/"TagsOriginal" para o "PorgentagemNormal" e "TagsGeradoFormatado"/"TagsOriginalFormatado" para o "PorcentagemFormatado". Para efeito de análise, foi considerado o "PorcentagemFormatado" conforme justificado anteriormente. As últimas linhas contém a média por coluna e a mediana por coluna.

Os projetos e os resultados podem ser conferidos em <https://nemo.inf.ufes.br/projetos/frameweb/>.

Tabela 11 – Resultados dos experimentos detalhados.

Projeto	Arquivo	TagsOriginal	TagsGerado	PorgentagemNormal	TagsOriginalFormatado	TagsGeradoFormatado	PorcentagemFormatado
Virtual Pool	Header.js	46	24	52.17%	26	12	46.15%
Virtual Pool	sinuca.js	25	15	60.0%	15	8	53.33%
Virtual Pool	jogo.js	21	10	47.62%	11	5	45.45%
Virtual Pool	rdf.js	21	9	42.86%	11	5	45.45%
Virtual Pool	perfil.js	67	25	37.31%	35	13	37.14%
Virtual Pool	entrar.js	25	7	28.0%	13	4	30.77%
Virtual Pool	index.js	20	7	35.0%	11	4	36.36%
Virtual Pool	ranking.js	31	15	48.39%	16	8	50.0%
Virtual Pool	jogar.js	30	17	56.67%	18	9	50.0%
Ufes Sports	footer.component.html	19	6	31.58%	11	3	27.27%
Ufes Sports	header.component.html	18	8	44.44%	9	4	44.44%
Ufes Sports	showevent.component.html	32	18	56.25%	16	9	56.25%
Ufes Sports	login.component.html	24	15	62.5%	13	12	92.31%
Ufes Sports	signup.component.html	40	15	37.5%	23	12	52.17%
Ufes Sports	config.component.html	4	2	50.0%	2	2	100.0%
Ufes Sports	student.component.html	32	22	68.75%	16	11	68.75%
Ufes Sports	sport.component.html	39	23	58.97%	20	13	65.0%
Ufes Sports	createevent.component.html	49	20	40.82%	27	15	55.56%
Ufes Sports	myevent.component.html	26	14	53.85%	13	7	53.85%
Ufes Sports	home.component.html	18	8	44.44%	9	4	44.44%
FeiranaMao	admin-produto.component.html	126	45	35.71%	66	30	45.45%
FeiranaMao	login.component.html	48	7	14.58%	27	5	18.52%
FeiranaMao	success.component.html	9	2	22.22%	5	1	20.0%
FeiranaMao	cart.component.html	54	29	53.7%	27	15	55.56%
FeiranaMao	cart-item.component.html	12	8	66.67%	6	4	66.67%
FeiranaMao	lojas.component.html	15	8	53.33%	8	4	50.0%
FeiranaMao	produto.component.html	24	12	50.0%	12	6	50.0%
FeiranaMao	dialog-produto-info.component.html	18	10	55.56%	9	5	55.56%
FeiranaMao	navbar.component.html	33	2	6.06%	17	1	5.88%
FeiranaMao	admin-loja.component.html	36	12	33.33%	20	9	45.0%
FeiranaMao	usuario.component.html	92	35	38.04%	48	21	43.75%
FeiranaMao	default.component.html	10	4	40.0%	5	2	40.0%
TasteUfes	Overlay.vue	8	6	75.0%	4	3	75.0%
TasteUfes	Footbar.vue	8	8	100.0%	4	4	100.0%
TasteUfes	Login.vue	39	29	74.36%	21	19	90.48%
TasteUfes	MenuMobile.vue	63	45	71.43%	32	23	71.88%
TasteUfes	MenuOption.vue	24	24	100.0%	12	12	100.0%
TasteUfes	HelpButton.vue	16	16	100.0%	8	8	100.0%
TasteUfes	UserCard.vue	54	38	70.37%	27	19	70.37%
TasteUfes	Toolbar.vue	32	21	65.62%	18	12	66.67%
TasteUfes	Snackbar.vue	8	8	100.0%	4	4	100.0%
TasteUfes	EditButton.vue	6	6	100.0%	3	3	100.0%
TasteUfes	DetailsButton.vue	6	6	100.0%	3	3	100.0%
TasteUfes	DeleteButton.vue	28	26	92.86%	14	13	92.86%
TasteUfes	NutritionFactsTable.vue	58	52	89.66%	31	26	83.87%
TasteUfes	RecipeData.vue	56	50	89.29%	30	25	83.33%
UMDB	Footer.js	6	4	66.67%	3	2	66.67%
UMDB	MovieSuggestion.js	16	9	56.25%	9	5	55.56%
UMDB	App.js	28	49	175.0%	25	25	100.0%
UMDB	SingleReview.js	11	7	63.64%	6	4	66.67%
UMDB	MovieInfo.js	43	17	39.53%	22	9	40.91%
UMDB	AdminInternalHeader.js	8	5	62.5%	4	3	75.0%
UMDB	MovieCard.js	14	16	114.29%	8	8	100.0%
UMDB	AdminContainer.js	9	9	100.0%	5	5	100.0%
UMDB	Header.js	12	6	50.0%	6	3	50.0%
UMDB	DefaultForm.js	10	8	80.0%	6	5	83.33%
UMDB	AdminSidebar.js	10	10	100.0%	5	5	100.0%
UMDB	Filter.js	33	31	93.94%	20	16	80.0%
UMDB	HomeContainer.js	14	14	100.0%	8	8	100.0%
UfesPay	profile.jsx	46	23	50.0%	26	14	53.85%
UfesPay	login.jsx	16	8	50.0%	9	5	55.56%
UfesPay	createacc.jsx	24	10	41.67%	14	7	50.0%
UfesPay	transferhistory.js	11	5	45.45%	6	3	50.0%
UfesPay	navtop.jsx	21	16	76.19%	12	8	66.67%
UfesPay	news.jsx	5	3	60.0%	3	2	66.67%
UfesPay	menuitem.js	8	8	100.0%	4	4	100.0%
UfesPay	layout.jsx	5	5	100.0%	3	3	100.0%
UfesPay	navitem.js	4	4	100.0%	2	2	100.0%
UfesPay	transferencia.js	6	2	33.33%	3	1	33.33%
UfesPay	menu.js	8	5	62.5%	4	4	100.0%
UfesPay	transfercard.jsx	49	43	87.76%	27	23	85.19%
UfesPay	landingpage.jsx	4	4	100.0%	3	3	100.0%
UfesPay	transfer.jsx	24	21	87.5%	15	14	93.33%
UfesPay	home.jsx	8	4	50.0%	5	3	60.0%
ClassiWeb	StyledButton.tsx	3	2	66.67%	2	1	50.0%
ClassiWeb	Footer.tsx	2	2	100.0%	1	1	100.0%
ClassiWeb	AdCard.tsx	38	39	102.63%	25	23	92.0%
ClassiWeb	MySelect.tsx	8	8	100.0%	4	4	100.0%
ClassiWeb	Categories.tsx	1	2	200.0%	1	1	100.0%
ClassiWeb	PageBase.tsx	8	7	87.5%	5	4	80.0%
ClassiWeb	Panel.tsx	31	27	87.1%	19	14	73.68%
ClassiWeb	Ads.tsx	44	35	79.55%	25	18	72.0%
ClassiWeb	Address.tsx	8	8	100.0%	5	5	100.0%
ClassiWeb	Carousel.tsx	8	10	125.0%	5	5	100.0%
ClassiWeb	ProductState.tsx	1	2	200.0%	1	1	100.0%
ClassiWeb	AppBar.tsx	43	28	65.12%	25	14	56.0%
ClassiWeb	CategoriesList.tsx	17	7	41.18%	11	4	36.36%
ClassiWeb	AdminPanel.tsx	11	10	90.91%	6	5	83.33%
ClassiWeb	CategoriesList.tsx	7	7	100.0%	4	4	100.0%
ClassiWeb	Ad.tsx	104	90	86.54%	58	47	81.03%
ClassiWeb	Feedback.tsx	8	4	50.0%	5	2	40.0%
Média	**Média**	**24.89**	**15.64**	**71.25%**	**13.64**	**8.62**	**69.04%**
Mediana	**Mediana**	**18**	**10**	**65.12%**	**11**	**5**	**66.67%**

Fonte: Produzido pelo autor.