



**UNIVERSIDADE FEDERAL DO CEARÁ**  
**CAMPUS CRATEÚS**  
**CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO**

**LUIS HENRIQUE CATUNDA RODRIGUES FARIAS**

**ESTUDO COMPARATIVO DA UTILIZAÇÃO DE DESIGN PATTERNS NO  
DESENVOLVIMENTO DE APLICAÇÃO WEB UTILIZANDO FRAMEWORKS  
FRONT-END**

**CRATEÚS**

**2021**

LUIS HENRIQUE CATUNDA RODRIGUES FARIAS

ESTUDO COMPARATIVO DA UTILIZAÇÃO DE DESIGN PATTERNS NO  
DESENVOLVIMENTO DE APLICAÇÃO WEB UTILIZANDO FRAMEWORKS  
FRONT-END

Trabalho de Conclusão de Curso apresentado ao  
Curso de Graduação em Sistemas de Informação  
do Campus Crateús da Universidade Federal  
do Ceará, como requisito parcial à obtenção do  
grau de bacharel em Sistemas de Informação.

Orientador: Prof. Me. Francisco Ander-  
son de Almada Gomes.

CRATEÚS

2021

Dados Internacionais de Catalogação na Publicação  
Universidade Federal do Ceará  
Biblioteca Universitária

Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

---

F238e Farias, Luis Henrique.

Estudo comparativo da utilização de design patterns no desenvolvimento de aplicação web utilizando frameworks front-end / Luis Henrique Farias. – 2021.  
85 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Crateús, Curso de Sistemas de Informação, Crateús, 2021.

Orientação: Prof. Me. Francisco Anderson de Almada Gomes.

1. Frameworks. 2. JavaScript. 3. Design Patterns. 4. Front-end. I. Título.

CDD 005

---

LUIS HENRIQUE CATUNDA RODRIGUES FARIAS

ESTUDO COMPARATIVO DA UTILIZAÇÃO DE DESIGN PATTERNS NO  
DESENVOLVIMENTO DE APLICAÇÃO WEB UTILIZANDO FRAMEWORKS  
FRONT-END

Trabalho de Conclusão de Curso apresentado ao  
Curso de Graduação em Sistemas de Informação  
do Campus Crateús da Universidade Federal  
do Ceará, como requisito parcial à obtenção do  
grau de bacharel em Sistemas de Informação.

Aprovada em:

BANCA EXAMINADORA

---

Prof. Me. Francisco Anderson de Almada  
Gomes (Orientador)  
Universidade Federal do Ceará (UFC)

---

Prof. Ma. Simone de Oliveira Santos  
Universidade Federal do Ceará (UFC)

---

Prof. Me. Ítalo Mendes da Silva Ribeiro  
Universidade Federal do Ceará (UFC)

---

Prof. Ma. Vitória Regina Nicolau Silvestre  
Universidade Federal do Ceará (UFC)

Dedico este trabalho a minha mãe e a toda minha família.

## **AGRADECIMENTOS**

Ao meu orientador Prof. Anderson Almada pela amizade, apoio e dedicação prestado para o desenvolvimento deste trabalho.

A todos os professores pelo conhecimento passado no decorrer da graduação, em especial a Prof. Lisieux Marie pelos conselhos e ensinamentos para meu desenvolvimento como pesquisador.

A minha mãe Salete e meu irmão Davi que sempre me incentivaram e apoiaram, antes e durante a minha vida acadêmica.

Aos meus amigos, pelo carinho e companheirismo que me proporcionaram ótimos momentos na universidade. Em especial ao William, Lara, João e Júnio, com quem convivi intensamente durante os últimos anos.

Os professores Ítalo Ribeiro, Simone Santos e Vitória Regina por aceitarem o convite de participação da banca e disponibilidade de tempo.

A Deus, por me permitir ultrapassar todos os obstáculos encontrados ao longo da graduação.

A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

“A vitalidade é demonstrada não apenas pela persistência, mas pela capacidade de começar de novo.”

(F.Scott Fitzgerald)

## RESUMO

As aplicações Web passaram por constantes mudanças nos últimos anos, representando uma evolução na abordagem convencional de desenvolvimento de software, a fim de atender as novas tendências e necessidades do mercado. Além disso, o desenvolvimento de grandes aplicações Web modernas possuem um crescimento constante de complexidade e vem se tornando cada vez mais comum. Nesse sentido, o mercado de desenvolvimento de software vem empregando *design patterns* e *frameworks* no desenvolvimento de aplicações, visando flexibilidade, desempenho e manutenibilidade. Este trabalho visa analisar e comparar *frameworks front-end JavaScript* (React e Vue.js) quando aplicados *design patterns* de software ao desenvolver aplicações Web, identificando fatores de influência e performance, para apresentar qual o *framework* é mais adequado neste contexto. Os resultados mostram que a viabilidade de aplicar *design pattern* em aplicações que utilizam *frameworks front-end* depende do ponto de vista avaliado, trazendo benefícios de sua aplicação quando comparado em tempo de resposta e consumo de memória. No entanto, se forem consideradas métricas de software foi verificado um impacto negativo, principalmente em termos de número de arquivos e complexidade computacional.

**Palavras-chave:** Frameworks. JavaScript. Design Patterns. Front-end.



## ABSTRACT

Web applications have gone through constant changes in recent years, representing an evolution in the conventional approach to software development to meet new market trends and needs. Furthermore, the development of large modern web applications has constantly grown in complexity, becoming more and more common. In this sense, the software development market has been employing *design patterns* and *frameworks* in application development, aiming at flexibility, performance, and maintainability. This work aims to analyze and compare *JavaScript front-end frameworks* (React and VueJS) when applying software *design patterns* when developing web applications, identifying influence and performance factors, to present what is the *framework* is more suitable in this context. The results show that the feasibility of applying *design pattern* in applications that use *front-end frameworks* depends on the evaluated point of view, bringing benefits to your application when compared in response time and memory consumption. However, if software metrics are considered, a negative impact was verified, mainly regarding the number of files and computational complexity.

**Keywords:** Frameworks. JavaScript. Design Patterns. Front-end.

## LISTA DE FIGURAS

Figura 1 – Etapas do Modelo cascata . . . . .	20
Figura 2 – Diagrama de classe do Modelo incremental . . . . .	21
Figura 3 – Diagrama de classe do padrão Singleton . . . . .	27
Figura 4 – Diagrama de classe do padrão Builder . . . . .	28
Figura 5 – Diagrama de classe do padrão factory method . . . . .	29
Figura 6 – Diagrama de classe do padrão Proxy . . . . .	31
Figura 7 – Diagrama do padrão <i>Adapter</i> - Adaptador de classe . . . . .	31
Figura 8 – Diagrama do padrão <i>Adapter</i> - Adaptador de objeto . . . . .	32
Figura 9 – Diagrama do padrão Iterator . . . . .	33
Figura 10 – Diagrama do padrão State . . . . .	34
Figura 11 – Relacionamento entre as camadas da arquitetura MVC . . . . .	35
Figura 12 – <i>Design patterns</i> na arquitetura MVC. . . . .	36
Figura 13 – Diagrama de classes do padrão arquitetural <i>Front Controller</i> . . . . .	37
Figura 14 – Diagrama de classe mostrando os relacionamentos para o padrão DAO . . . . .	38
Figura 15 – <i>Frameworks Web</i> com o maior numero de questões no Stack Overflow . . . . .	41
Figura 16 – Árvore de componentes do React, compara as diferenças entre o DOM virtual e o DOM real . . . . .	42
Figura 17 – Interface visual da aplicação, abstraída por uma árvore de componentes . . . . .	42
Figura 18 – Os elementos básicos de uma aplicação Angular . . . . .	44
Figura 19 – Procedimentos Metodológicos . . . . .	49
Figura 20 – Diagrama de Casos de Uso . . . . .	52
Figura 21 – Diagrama de classes dos protótipos . . . . .	53
Figura 22 – Diagrama de componentes . . . . .	54
Figura 23 – Diagrama de atividade . . . . .	54
Figura 24 – Tela de login . . . . .	55
Figura 25 – Tela inicial . . . . .	55
Figura 26 – Tela de cadastro de atividade . . . . .	56
Figura 27 – Tela de atualizar atividade . . . . .	56
Figura 28 – Tela de deletar atividade . . . . .	57
Figura 29 – Camadas da Arquitetura de Software. . . . .	58
Figura 30 – Relacionamento DAO. . . . .	58

Figura 31 – Relacionamento Builder. . . . .	59
Figura 32 – Relacionamento Singleton. . . . .	59
Figura 33 – Resultado dos testes para consumo de memória . . . . .	64
Figura 34 – Resultado dos testes para consumo de CPU. . . . .	65
Figura 35 – Resultado dos testes para Tempo de renderização. . . . .	65
Figura 36 – Resultado dos testes para Tempo de resposta - Login de Usuário. . . . .	66
Figura 37 – Resultado dos testes para Tempo de resposta - Cadastro de Usuário. . . . .	66
Figura 38 – Resultado dos testes para Tempo de resposta - Cadastro de atividade. . . . .	67
Figura 39 – Resultado dos testes para Tempo de resposta - Atualizar atividade. . . . .	67
Figura 40 – Resultado dos testes para Tempo de resposta - Excluir atividade. . . . .	68

## LISTA DE TABELAS

Tabela 1 – Dados dos <i>Frameworks</i> no <i>Github</i> . . . . .	40
Tabela 2 – Comparativo Entre os Trabalhos Relacionados . . . . .	48
Tabela 3 – Descrição de Requisitos Funcionais . . . . .	52
Tabela 4 – Descrição de Requisitos Não Funcionais . . . . .	52
Tabela 5 – Métricas de software . . . . .	60
Tabela 6 – Métricas de desempenho . . . . .	60
Tabela 7 – Métricas de software sem design pattern . . . . .	62
Tabela 8 – Métricas de software com design pattern . . . . .	63
Tabela 9 – Tempo de resposta comparando a versão sem e com padrão . . . . .	68
Tabela 10 – Comparativo Entre os Trabalhos Relacionados e a Proposta . . . . .	69

## LISTA DE ABREVIATURAS E SIGLAS

AJAX	Asynchronous Javascript And XML
BD	Banco de dados
CPU	Central Processing Unit
CRUD	Create (Criação), Read (Consulta), Update (Atualização) e Delete (Destruição)
CSS	Cascading Style Sheets
DAO	Data Access Object
DOM	Document Object Model
GoF	Gang of Four
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
MVC	Model-View-Controller
PHP	Hypertext Preprocessor
RAM	Ram Truck Division
REST	Representational State Transfer
SPA	Single Page Application. Arquitetura de aplicações WEB de uma página
UI	User Interface
URL	Uniform Resource Locator (Localizador Uniforme de Recursos)
Web	World Wide Web
XML	Extensible Markup Language

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	<b>15</b>
<b>1.1</b>	<b>Contextualização</b>	<b>15</b>
<b>1.2</b>	<b>Justificativa</b>	<b>15</b>
<b>1.3</b>	<b>Objetivos</b>	<b>16</b>
<b>1.3.1</b>	<b>Objetivo Geral</b>	<b>16</b>
<b>1.3.2</b>	<b>Objetivos Específicos</b>	<b>16</b>
<b>1.4</b>	<b>Estrutura do documento</b>	<b>17</b>
<b>2</b>	<b>FUNDAMENTAÇÃO TEÓRICA</b>	<b>18</b>
<b>2.1</b>	<b>Engenharia de Software</b>	<b>18</b>
<b>2.1.1</b>	<b>Modelos de processo de software</b>	<b>19</b>
2.1.1.1	<i>Modelo cascata</i>	19
2.1.1.2	<i>Modelo Incremental</i>	21
<b>2.1.2</b>	<b>Desenvolvimento Ágil de Software</b>	<b>22</b>
2.1.2.1	<i>Scrum</i>	23
<b>2.2</b>	<b>Desenvolvimento de Software para Web</b>	<b>23</b>
<b>2.3</b>	<b>Design Patterns</b>	<b>25</b>
<b>2.3.1</b>	<b>Padrões Criacionais</b>	<b>27</b>
2.3.1.1	<i>Singleton</i>	27
2.3.1.2	<i>Builder</i>	28
2.3.1.3	<i>FactoryMethod</i>	29
<b>2.3.2</b>	<b>Padrões Estruturais</b>	<b>30</b>
2.3.2.1	<i>Proxy</i>	30
2.3.2.2	<i>Adapter</i>	31
<b>2.3.3</b>	<b>Padrões Comportamentais</b>	<b>32</b>
2.3.3.1	<i>Iterator</i>	32
2.3.3.2	<i>State</i>	33
<b>2.3.4</b>	<b>Arquiteturas e Padrões Web</b>	<b>34</b>
2.3.4.1	<i>MVC - Model-View-Controller</i>	35
2.3.4.2	<i>Front Controller</i>	36
2.3.4.3	<i>DAO - Data Access Object</i>	38

2.4	Frameworks Web JavaScript . . . . .	39
2.4.1	<i>React</i> . . . . .	40
2.4.2	<i>Vue.js</i> . . . . .	41
2.4.3	<i>Angular</i> . . . . .	43
2.5	Considerações Finais . . . . .	44
3	TRABALHOS RELACIONADOS . . . . .	45
3.1	Trabalho de Cechinel <i>et al.</i> . . . . .	45
3.2	Trabalho de Mantoan . . . . .	46
3.3	Trabalho de Thung <i>et al</i> . . . . .	47
3.4	Comparativo Entre os Trabalhos . . . . .	47
3.5	Considerações Finais . . . . .	48
4	METODOLOGIA . . . . .	49
4.1	Considerações Finais . . . . .	50
5	PROPOSTA . . . . .	51
5.1	Frameworks Front-End Web . . . . .	51
5.2	Prova de Conceito . . . . .	51
5.3	Design Patterns . . . . .	57
5.4	Métricas de Desempenho e Software . . . . .	59
5.5	Considerações Finais . . . . .	60
6	RESULTADOS . . . . .	61
6.1	Plano de Testes . . . . .	61
6.2	Resultados Encontrados . . . . .	62
6.2.1	<i>Métricas de Software</i> . . . . .	62
6.2.2	<i>Métricas de Desempenho</i> . . . . .	63
6.3	Comparativo Entre os Trabalhos . . . . .	68
6.4	Considerações Finais . . . . .	69
7	CONCLUSÃO . . . . .	70
7.1	Limitações . . . . .	71
7.2	Trabalhos Futuros . . . . .	71
	REFERÊNCIAS . . . . .	73
	APÊNDICE A – Implementação da Prova de Conceito . . . . .	76

# 1 INTRODUÇÃO

## 1.1 Contextualização

Aplicações Web estão cada vez mais presente no nosso cotidiano, possuindo um crescimento constante de desenvolvimento que engloba uma boa parte da produção de empresas de softwares (ROPELATO, 2007). Com essas aplicações, existe uma evolução da abordagem convencional de software, mantendo os princípios de engenharia para alcançar aplicações de qualidade. Essas aplicações seguem os requisitos dos usuários, com o desenvolvimento considerando a infra-estrutura Web para o processamento e disponibilização (CONTE *et al.*, 2005; GERALDES; MIRANDA, 2002).

Normalmente, o desenvolvimento Web assume duas abordagens: aplicações hiper-mídia Web ou aplicações de software Web. Na hiper-mídia Web, a informação é caracterizada por utilizar interfaces direcionadas para a navegação, disponibilizada através da Web. Já uma aplicação de software Web são aplicações convencionais que requerem da infra-estrutura Web para o processamento (CONTE *et al.*, 2005; THUNG *et al.*, 2010). O termo aplicação Web representa a combinação da duas abordagens.

## 1.2 Justificativa

O mercado de desenvolvimento de software está em constante evolução e tem passado por profundas mudanças de paradigmas e tendências tecnológicas, influenciando a maneira como são implementadas aplicações Web<sup>1</sup>.

A fim de atender a essas novas necessidades do mercado, as empresas vêm empregando *design patterns* e *frameworks* Web (e.g., React.js<sup>2</sup>, Vue.js<sup>3</sup>) no desenvolvimento de suas aplicações (LEFF; RAYFIELD, 2001). Os *design patterns* descrevem soluções para problemas comuns que ocorrem com frequência no desenvolvimento de software. Eles aplicam o reuso de soluções genéricas, aumentando a produtividade e reduzindo o retrabalho (MALDONADO *et al.*, 2002).

---

<sup>1</sup> <<https://blog.revelo.com.br/desenvolvedor-de-software-7-desafios-para-sua-carreira/>>

<sup>2</sup> <<https://reactjs.org/>>

<sup>3</sup> <<https://vuejs.org/>>



Dessa forma, é possível ter mais flexibilidade, qualidade e entregas em prazos cada vez mais curtos, o que torna a utilização de *design patterns* e *frameworks* Web essenciais no contexto de desenvolvimento (GORLA; FOSCHINI, 2014; MORALES-CHAPARRO *et al.*, 2007). No entanto, a implementação do *design patterns* pode ser difícil, principalmente devido a curva de aprendizagem e torna-se custosa em prazos e custos, se adotado de forma tardia na aplicação (VORA, 2009). Além disso, a maioria das abordagens de padrão, especialmente aquelas que contam com polimorfismo, são ligeiramente mais lentas do que uma solução equivalente que não incentiva a reutilização<sup>4</sup>.

Durante as pesquisas realizadas nesse trabalho, foi verificado a inexistência de trabalhos na literatura que avaliam o desempenho computacional dos *frameworks* Web atuais do mercado ou qualquer avaliação a nível de métricas de software no desenvolvimento de uma aplicação Web quando aplicado *design patterns*.

### 1.3 Objetivos

O principal objetivo desse trabalho é realizar uma pesquisa voltada para uma comparação da viabilidade da aplicação de *design patterns* em *framework front-end* Web.

#### 1.3.1 Objetivo Geral

Estudo comparativo entre *frameworks front-end* Web quando aplicados *design patterns* no desenvolvimento de aplicações, visando encontrar o *framework* com melhor desempenho computacional e menor impacto da utilização do *design patterns* no que diz respeito a métricas de software.

#### 1.3.2 Objetivos Específicos

Os seguintes objetivos específicos foram estabelecidos para o trabalho:

- Apresentar a viabilidade da utilização de *design patterns* em dois *frameworks front-end* Web (Vue.js e React.js) a partir da implementação de uma aplicação Web;
- Avaliar a utilização de *design patterns* com relação a métricas de desempenho

---

<sup>4</sup> <<https://www.gofpatterns.com/design-patterns/module7/limitations-of-design-patterns.php>>

computacional;

- Avaliar a utilização de *design patterns* com relação a métricas de software; e
- Apresentar o *framework front-end* Web com melhor desempenho computacional e menor impacto com relação as métricas de software quando aplicado *design patterns*;

#### 1.4 Estrutura do documento

Este trabalho está organizado em sete capítulos, conforme descrito a seguir:

- **Capítulo 2** - Pesquisa bibliográfica sobre o campo de interesse da pesquisa, a fim de compreender os conceitos e elementos fundamentais do trabalho: Engenharia de Software, Desenvolvimento Web, *Design Patterns* e *Frameworks front-end* Web;
- **Capítulo 3** - Análise e comparação de pesquisas que estão relacionadas ao presente trabalho;
- **Capítulo 4** - Apresentação da metodologia proposta para a realização da pesquisa, exibindo a distribuição das etapas adotadas.
- **Capítulo 5** - Apresentação da proposta e o conjunto de procedimentos para o desenvolvimento do trabalho. Além disso, a apresentação da aplicação Web desenvolvida com a utilização de *design pattern* e os *frameworks front-end* Web.
- **Capítulo 6** - Apresentação dos experimentos que foram utilizados para avaliar as versões da aplicação Web com *design patterns* e *frameworks front-end* Web. Além disso, são apresentados os resultados encontrados no trabalho proposto.
- **Capítulo 7** - Por fim, são apresentadas as conclusões e os trabalhos futuros do trabalho proposto.

## 2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo fornece os conceitos fundamentais e uma visão geral dos *design patterns* para o desenvolvimento de aplicações abordados na presente pesquisa. A seção 2.1 apresenta os conceitos de Engenharia de Software, os processos de desenvolvimentos e seus diversos modelos. A seção 2.2 aborda a evolução dos processos de desenvolvimento de Software para Web. A seção 2.3 aponta os diversos *Design Patterns* e as arquiteturas. Por fim a seção 2.4, define o que é um *framework* e apresenta alguns *frameworks front-end Web*.

### 2.1 Engenharia de Software

A Engenharia de Software é uma abordagem sistemática e disciplinada para o desenvolvimento de software, o qual envolve processos técnicos para atender os aspectos da produção de software, desde especificações do projeto até sua manutenção (PRESSMAN, 2006). Além disso, ela também foca no gerenciamento de projeto, desenvolvimento de ferramentas e métodos para apoiar a produção de software de alta qualidade (SOMMERVILLE, 2011). Para Keyes (2002), a Engenharia de Software são metodologias e ferramentas de desenvolvimento que identificam as diferentes etapas ao desenvolver um sistema, incluindo requisitos de software, estudo de viabilidade, especificação de design e um plano de teste completo, a fim de implementar sistemas satisfatórios aos usuários finais.

Segundo Pressman (2006), a engenharia de software reúne princípios da engenharia, a fim de entregar economicamente um software confiável e funcional em máquinas reais. Ela é composta de três elementos fundamentais (metodologias, ferramentas e procedimentos) que auxiliam o controle e planejamento do desenvolvimento de software, além da efetivação de qualidade. Conforme contextualizado por Casteleyn *et al.* (2009), o desenvolvimento de software é um processo de inovação de soluções de software para produtos e serviços.

No desenvolvimento de software existem diferentes tipos de processos, mas todos devem incluir as seguintes atividades fundamentais (SOMMERVILLE, 2011):

- **Especificação de software:** identificar e mapear as funcionalidades do software e suas restrições;
- **Projeto e implementação de software:** desenvolvimento do software seguindo

suas especificações;

- **Validação de software:** verificar que o software atende a todos os requisitos do cliente; e
- **Evolução de software:** o software deve ser aprimorado para atender novas necessidades do cliente.

O desenvolvimento dos processos de software podem ser aprimorados para promover a maior qualidade e/ou reduzir seu custo e tempo. Desta forma, a seguir, serão apresentados alguns modelos de processos de software.

### **2.1.1 Modelos de processo de software**

No desenvolvimento de software é importante definir a sequência em que as atividades do processo serão realizadas. Diferentes modelos de processo (ou paradigmas de engenharia de software) foram desenvolvidos para fornecer uma estrutura útil de trabalho. Cada modelo representa uma abordagem usada para a criação do software. Além de integrar os *stakeholders* a desempenhar um papel em relação a esse processo, existe uma garantia de estabilidade, controle e organização. Os modelos de processo que serão abordados na presente pesquisa são o modelo cascata e incremental, apresentados nas próximas seções.

#### **2.1.1.1 Modelo cascata**

Para Pressman (2011), modelo cascata é um modelo de desenvolvimento de software sequencial (ilustrado na figura 1), na qual o processo se inicia na fase de requisitos e análise, para então avançar ao longo do Projeto, Implementação, Integração, Operação e Manutenção. Conforme definido por Sommerville (2011), o modelo é um exemplo de processo dirigido a planos, onde as atividades devem ser planejadas e definidas antes da execução.

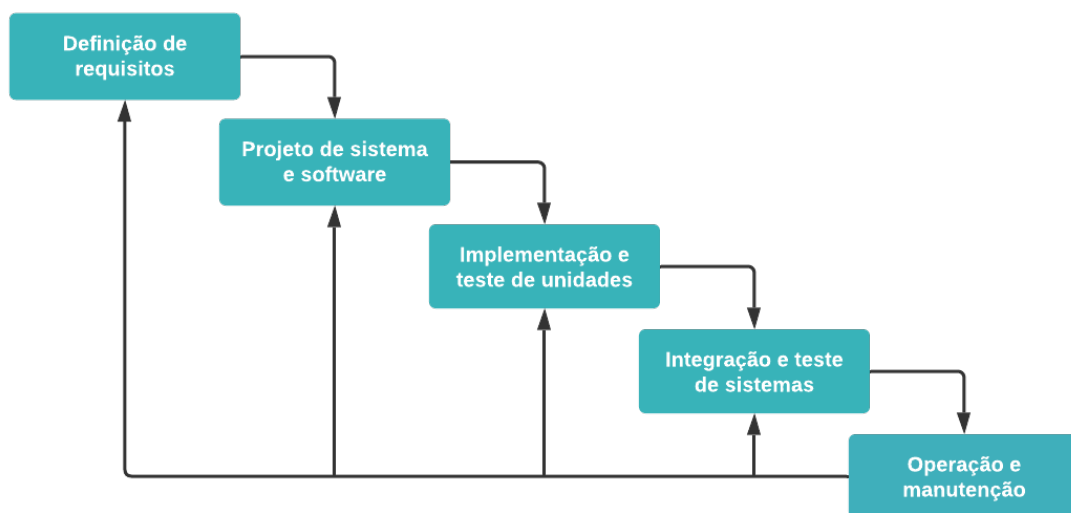
As atividades fundamentais no desenvolvimento com modelo cascata são:

- **Análise e definição de requisitos:** especifica as necessidades, restrições e funções identificadas por meio de consultas aos usuários;
- **Projeto de sistema e software:** define a arquitetura do sistema, envolve a especificação de subsistemas do software e seus relacionamentos;
- **Implementação e teste de unidades:** o processo de desenvolvimento e codi-

ficação do software. O teste unitário certifica (valida) que as implementações atendem as especificações;

- **Integração e teste de sistemas:** integra todas as unidades do sistema e testa como um sistema completo, para garantir que todos os requisitos do sistema foram atendidos. Após o teste, o software é entregue ao cliente; e
- **Operação e manutenção:** o sistema é instalado e começa a operar. A manutenção vai corrigir erros, adicionar novas funções não identificadas anteriormente e aperfeiçoar o sistema (evolução).

Figura 1 – Etapas do Modelo cascata



Fonte – Adaptado de Sommerville (2011)

Esse modelo recebeu diversas críticas, o qual questionam se ele deve ser utilizado em todas as situações (PRESSMAN, 2006). Os principais problemas encontrados na aplicação do modelo são:

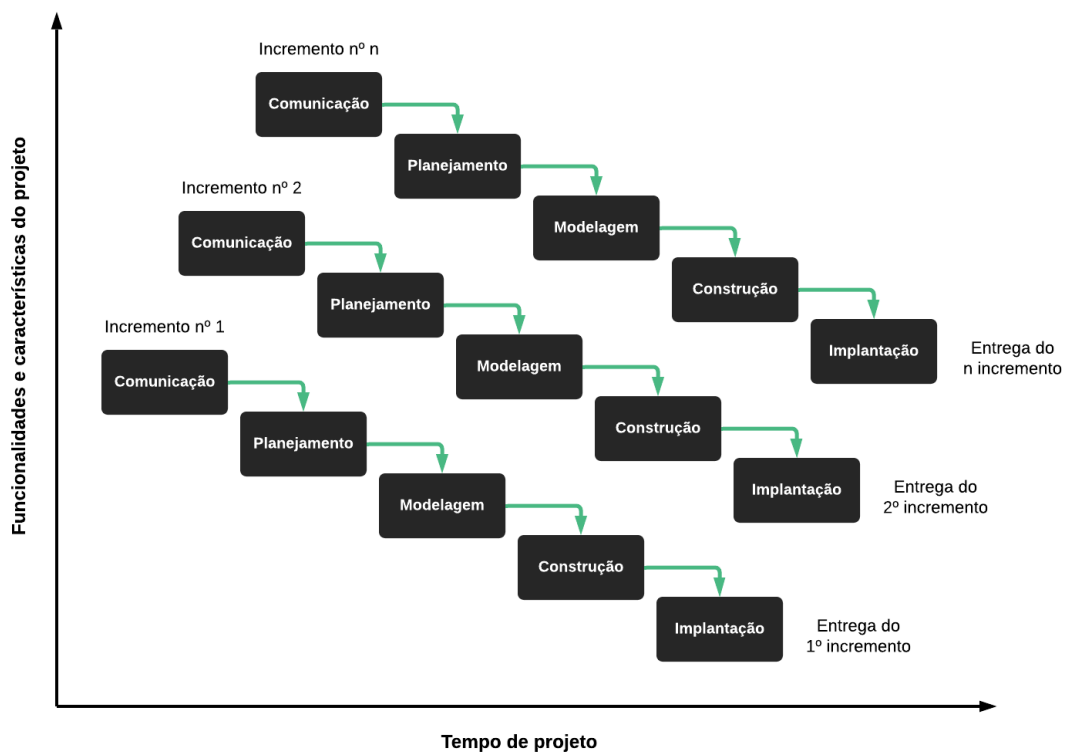
1. O fluxo sequencial raramente é seguido pelos projetos reais;
2. Dificuldade do cliente estabelecer os requisitos explicitamente; e
3. O tempo de entrega de uma versão para o cliente, que só é disponibilizada no fim da modelagem (requisitos e análise).

O principal problema do modelo é a inflexibilidade de adaptação, é difícil realizar ações retroativas para corrigir erros cometidos em atividades já concluídas. A seguir, será apresentado o modelo incremental que incorpora características do modelo cascata.

### 2.1.1.2 Modelo Incremental

O Modelo Incremental surge como uma melhoria do Modelo em Cascata. Ao invés de especificar e desenvolver tudo de uma só vez, este modelo trabalha com pequenos pedaços (incrementos) de software entregues um de cada vez, desenvolvido de forma linear, conforme figura 2. Esse modelo combina elementos do Modelo em Cascata aplicados de maneira iterativa, ou seja, de forma que o progresso aconteça através de novos incrementos, melhorados a cada novo resultado apresentado (PRESSMAN, 2006).

Figura 2 – Diagrama de classe do Modelo incremental



Fonte – Adaptado de Pressman (2006).

Esse modelo é muito útil quando não há mão-de-obra disponível no momento para uma implementação completa, dentro do prazo estipulado de entrega no projeto. Esse método de desenvolvimento vai ser mais aprofundado no desenvolvimento Ágil, apontado nas seções seguintes, tratando melhor esse desenvolvimento de partes de software a cada ciclo.

### 2.1.2 *Desenvolvimento Ágil de Software*

As metodologias ágeis constituem uma nova metodologia de produção de software criada para suprir a pressão do mercado sobre processos mais leves e flexíveis, com ciclos cada vez mais curtos de desenvolvimento. Segundo Pontes e Arthaud (2018), a premissa das metodologias ágeis é a adaptação a novos fatores decorrentes do desenvolvimento do projeto, em que é previsto mudanças. Já metodologias tradicionais preveem detalhes do processo de desenvolvimento previamente e acabam tendo problemas, quando são necessárias mudanças, como retrabalho e atrasos na entrega do projeto.

O termo Metodologias Ágeis tornou-se popular em 2001, com um encontro de especialistas de processos de software representando diferentes métodos ágeis, a fim de discutir princípios comuns compartilhados por todos esses métodos para a obtenção de bons resultados. Dessa reunião surgiu a aliança ágil e estabeleceu o “Manifesto Ágil”. Os conceitos chave deste manifesto são (BECK *et al.*, 2001):

- Indivíduos e interações mais que processos e ferramentas;
- Software executável mais que documentação;
- Colaboração do cliente mais que negociação de contratos;
- Respostas rápidas a mudanças mais que seguir planos.

O Manifesto Ágil, segundo Filho (2008), ressalta que o elemento de mais valor para as metodologias ágeis é saber lidar com pessoas, na colaboração com o cliente para encontrar soluções eficientes e na resposta rápida às mudanças para entregar software com qualidade. No manifesto não é previsto a rejeição aos processos e ferramentas, a documentação, a negociação de contratos ou o planejamento, mas simplesmente mostra que eles têm importância secundária.

Dentre várias metodologias presentes na literatura, este trabalho proposto utilizará o *Scrum*, pois facilitará a gestão do projeto e desenvolvimento do software. Ele possibilita entregar a aplicação de forma mais rápida, ao focar nessa abordagem de entregas, avaliando seus custos e melhorias (FRANCO, 2007).

### 2.1.2.1 *Scrum*

*Scrum* é uma metodologia criada na década de 1990 por Jeff Sutherland e explorada por Takeuchi e Nonaka (1986) no artigo “The new product development game”, no qual descrevem que equipes pequenas, multidisciplinares e auto gerenciáveis produzem melhores resultados.

O *Scrum* não aborda uma técnica específica para a etapa de implementação no desenvolvimento de software, ele se concentra no gerenciamento do projeto, orientando o trabalho da equipe para desenvolver sistemas ágeis e flexíveis, em ambientes adaptativos (FRANCO, 2007). Além disso, o *Scrum* faz uso de abordagem de gestão de projeto, iterativo e incremental aplicada através de três papéis principais (SCHWABER, 2004):

- **Product Owner:** responsável por guiar o projeto, seguindo os interesses do cliente e das demais partes interessadas no projeto;
- **Time:** desenvolver funcionalidades do projeto, transformar as atividades que precisam ser feitas em incrementos funcionais;
- **Scrum Master:** garantir que a equipe siga as regras e práticas do *Scrum*, além de atuar como facilitador, responsável por remover impedimentos do projeto.

O método se destaca pelo desenvolvimento ágil de produtos complexos, reúne atividades de monitoramentos e *feedback*, reuniões rápidas e frequentes com toda a equipe, para alinhar e sinalizar obstáculos e priorizar o trabalho a ser implementado na *Sprint*, que é um período de no máximo de trinta dias, o qual é planejado a realização das atividades do *backlog*, que guarda as atividades a serem realizadas (BALLE, 2011). Consequentemente, o *Scrum* traz mais agilidade no desenvolvimento de Software para Web, flexibilizando o planejamento, e possibilita entregar a aplicação de forma mais rápida ao cliente.

## 2.2 Desenvolvimento de Software para Web

A World Wide Web surgiu em 1989 por meio de Tim Berners-Lee no CERN (Organização Europeia para a Pesquisa Nuclear), que propôs um novo protocolo, um integrador de informações para a internet. O novo sistema projetado permite que informações fossem facilmente acessadas, em qualquer lugar, conectadas através da internet e apresentadas em formato de



hipertexto (HTML) (CECHINEL *et al.*, 2017). A primeira página na internet foi lançada em 1990 por Tim Berner-Lee, hospedada no servidor de rede também criado pelo cientista (ROPELATO, 2007).

A Web passou por diversas transformações durante o tempo, o qual pode ser dividida em três / quatro ondas: Web apenas de leitura, Web de leitura e escrita e a Web programável e a próxima era, já denominada Web simbiótica (CHOUDHURY, 2014).

A versão Web 1.0 inventada por Tim Burners-Lee, foi identificada como a Web somente leitura. Permite que as aplicações sejam capazes de fornecer informações estáticas, limitadas em questão de comunicação e a interação com os leitores. Realizando a transferência de bens e conhecimentos (KULESZA *et al.*, 2018). Apesar da Web 1.0 suportar uma variedade de protocolos da Web, o mais comum utilizado era o HTTP, por fornecer um padrão de comunicação entre os navegadores e servidores Web (CECHINEL *et al.*, 2017).

A segunda onda, chamada de Web 2.0 é conhecida como Web social ou Web de leitura e escrita desenvolvida por Dale Dougherty. Não é um avanço específico de tecnologia, e sim um conjunto de padrões de design e arquiteturas. Migrando os negócios para a internet como plataformas. Focam nas características de interação e participação entre comunidades, pessoas e software, em especial nas aplicações baseadas em Web (CECHINEL *et al.*, 2017). Tornando o ambiente mais dinâmico para que haja a participação e contribuição dos usuários para a organização de conteúdo.

Com o aumento de acesso e pessoas conectadas na Web, a complexidade de conteúdo servidos na Web aumentou, fazendo com que conteúdos estáticos se comportassem como aplicações de forma similar a aplicações *desktop*. Em 1995 foi apresentado o *Javascript*, uma linguagem de *script* que permite criar aplicações Web mais dinâmicas.

A tecnologia fundamental para a Web 2.0 é o AJAX que permite que aplicações Web possam enviar e solicitar dados de um servidor de forma assíncrona, sem a necessidade de atualizar a página existente (KULESZA *et al.*, 2018). Além do REST, sendo um estilo de arquitetura de software para a criação de serviços Web que baseia-se nas operações do método HTTP (GET, POST, PUT e DELETE) para criar aplicativos interativos (BOMFIM; SAMPAIO, 2008).

Com a consolidação do HTML5 e das ferramentas de implementação de interface gráfica, surgiram as SPA, aplicações que recuperam todos os códigos HTML, *JavaScript* e CSS necessários em um único carregamento de página (KULESZA *et al.*, 2018). A linguagem *JavaScript* evoluiu bastante ao longo dos anos, possibilitando aos desenvolvedores atuais várias alternativas de *frameworks JavaScript*, como React, Angular e Vue.js.

A terceira onda, chamada de Web 3.0 foi desenvolvida por John Markoff em 2006, identificada como a “Web executável”, tendo como ponto forte a facilidade de desenvolver sistemas completos na própria Web e a otimização da experiência online, o qual foi alavancado pela computação em Nuvem (KULESZA *et al.*, 2018).

A quarta onda, chamada de Web 4.0, também conhecida como a Web simbiótica, tem como objetivo a interação simbiótica entre o homem e a máquina (AGHAEI *et al.*, 2012). Desenvolvimento de telecomunicações avançadas, a partir de interfaces controladas usando Web 4.0, ou seja, as máquinas seriam inteligentes ao ler o conteúdos da Web, identificando as melhores formas de executar e decidir passos para carregar sites com qualidade, desempenho e construir interfaces de comando (CHOUDHURY, 2014).

A evolução na Web mudou radicalmente o desenvolvimento de aplicações Web, passando de páginas estáticas para um estágio de Web executável. A seguir, serão apresentados diferentes padrões de projeto que oferecem benefícios práticos no desenvolvimentos das aplicações.

### **2.3 Design Patterns**

Os desenvolvedores de software, possivelmente, enfrentam problemas específicos que já foram encontrados e resolvidos em projetos anteriores, mas a falta de documentação de problemas e soluções de projeto, impossibilitam aproveitar experiências passadas. Os *design patterns* solucionam esses problemas.

O conceito de padrões foi primeiramente apresentado por Christopher Alexander Ishikawa *et al.* (1977), no livro: Uma Linguagem de Padrões, descrevendo no contexto para projeto de ambiente urbano (Arquitetura). Porém os profissionais de Software incorporaram a ideia, aplicando o conceito na criação de *design patterns* para desenvolvimento de software.

Os *design patterns* identificam aspectos particulares de uma estrutura de projeto, a fim de reutilizá-la para a criação de projetos orientado a objetos, abordando seus papéis, colaborações e suas responsabilidades. Gamma *et al.* (2000) definem que *design patterns* “são descrições de objetos e classes comunicantes que precisam ser personalizadas para resolver um problema geral de projeto num contexto particular”.

Gamma *et al.* (2000) apresentam um formato unificado de um *design patterns*, constituído com as seguintes características:

- **Nome:** definir um nome que expresse sua essência;
- **Intenção:** descreve questões do propósito desse *design patterns*;
- **Motivação:** um cenário que descreve um problema de projeto que o padrão solucione;
- **Aplicabilidade:** informa as situações nas quais o *design patterns* pode ser aplicado;
- **Estrutura:** uma representação gráfica de diagramas de classe usando (OMT) e diagramas de interação para ilustrar as classes e suas relações entre objetos;
- **Participantes:** As classes e/ou objetos que participam e quais as suas responsabilidades;
- **Colaborações:** como as classes participantes colaboram para realizar suas tarefas;
- **Consequências:** quais as vantagens e desvantagens do padrão, avaliar os resultados alcançados da sua aplicação;
- **Implementação:** quais aspectos preciso conhecer para implementar o padrão e se tem especificações de linguagem;
- **Exemplo de código:** códigos que ilustram como o desenvolvedor pode implementar o padrão;
- **Usos conhecidos:** exemplos de casos de sistemas reais que o padrão foi utilizado, pelo menos dois domínios distintos;
- **Padrões relacionados:** relacionar os padrões existentes, ver quais são similares, diferenças e que devem ser usados em conjunto.

Conforme descrevem Gamma *et al.* (2000), os *design patterns* podem ser categorizados por dois critérios: propósito e intenção. O propósito é dividido em três categorias principais:

criacional, estrutural e comportamental.

### 2.3.1 Padrões Criacionais

Os padrões criacionais fornecem mecanismos ao processo de criação de objetos, deixando seu código mais flexível e reutilizável. Auxilia em tornar o sistema independente da forma que seus objetos são criados, compostos e representados. Um padrão de criação de classe emprega herança para alternar a classe instanciada. No entanto, padrões criacionais voltados a objetos, atribui outro objeto para o processo de criação (GAMMA *et al.*, 2000). A seguir, serão apresentados os padrões *Singleton*, *Builder* e *Factory Method*.

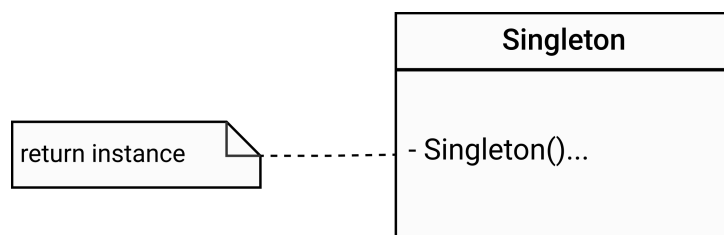
#### 2.3.1.1 Singleton

Conforme Gamma *et al.* (2000), o padrão *Singleton* assegura que uma instância única de objeto seja criada em uma classe e prover um ponto de acesso global para a mesma em toda aplicação. Algumas classes têm a necessidade de assegurar a existência de uma única instância de objeto com um tipo específico e ser acessível para diferentes objetos. O padrão *Singleton* vai garantir que nenhum outro objeto exista (PEREIRA, 2008).

A utilização do padrão *Singleton* é indicado quando houver a necessidade de apenas uma instância de objeto disponível para todos os clientes e quando precisa haver um controle mais restrito para as variáveis globais, onde o cliente utiliza essa instância estendida sem alterar o código (GAMMA *et al.*, 2000).

Como exemplo demonstrado na figura 3, é declarado um método público *getInstance()*, responsável por retornar a instância da classe a qual pertence. O construtor da classe é o método privado *Singleton()*, onde somente o *getInstance()* vai obter o objeto *singleton*.

Figura 3 – Diagrama de classe do padrão Singleton



Fonte – Adaptado de Lino (2011)

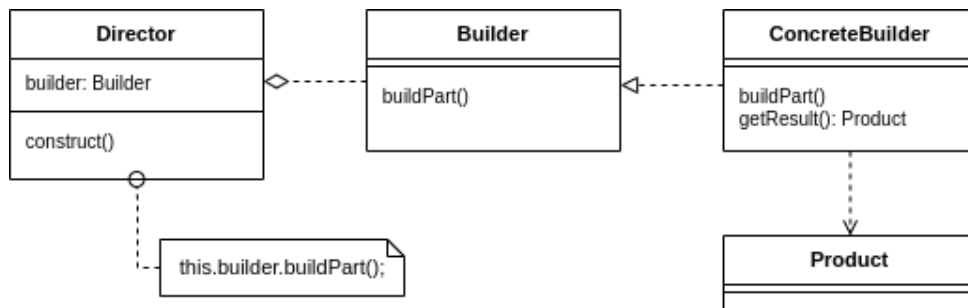
O padrão apresenta um controle de acesso de instância única dentro da própria classe, onde o cliente não se preocupa com a existência ou não de uma instância no singleton (SHALLOWAY; TROTT, 2004). A seguir, será apresentado o padrão Builder que incorpora outro mecanismo dos padrões criacionais.

### 2.3.1.2 Builder

O padrão Builder permite a construção de objetos complexos sem que precisamos conhecer sua representação, de modo que o mesmo processo de construção possa criar diferentes representações.

O Builder deve ser utilizado quando deseja-se a construção de objetos passo a passo, o mesmo código precisa produzir diferentes representações do mesmo produto usando apenas as etapas necessárias que o compõem. Após implementar o padrão, o algoritmo vai ser capaz de criar todas as representações do produto disponíveis (GAMMA *et al.*, 2000).

Figura 4 – Diagrama de classe do padrão Builder



Fonte – Adaptado de Gamma *et al.* (2000).

A estrutura do padrão Builder é ilustrada na figura 4, representando seus elementos de construção de objetos, onde temos um classe Director que chama o método de construção da classe Builder, que declara as etapas para a produção do objeto-produto, a classe ConcreteBuilder define e mantém diferentes implementações para construir a classe Product.

Segundo Gamma *et al.* (2000), o uso do padrão Builder permite a construção em etapas e a reutilização do mesmo código para criar diferentes representações, abstraindo seu código de criação. Porém a complexidade geral do código é aumentada, ao exigir criar multiplas novas classes.

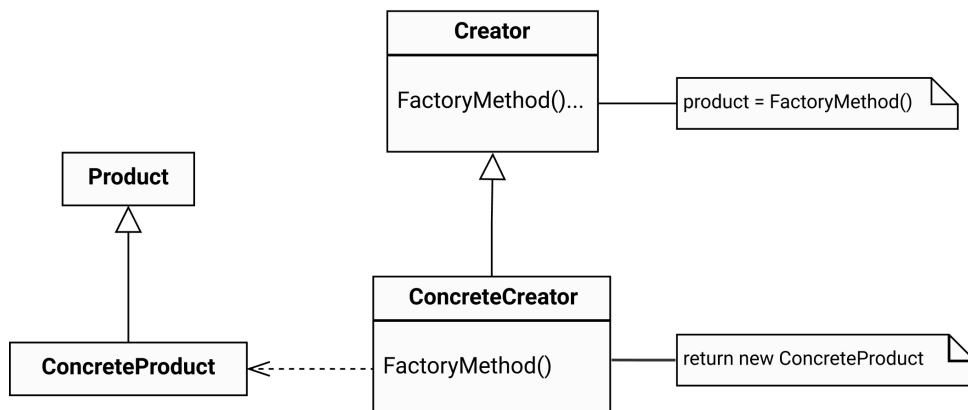
### 2.3.1.3 FactoryMethod

O *design pattern Factory Method* fornece uma interface com mecanismos para criar objetos, mas permite que as subclasses decidam qual a classe instanciar (SHALLOWAY; TROTT, 2004). Os *frameworks* são frequentemente responsáveis pela criação de objetos, eles usam classes abstratas para definir e manter a relação entre esses objetos. Considerando que uma classe abstrata do *framework* precisa instanciar uma derivação de outra classe, ela não consegue prever a subclasse a ser instanciada. O padrão soluciona esse problema ao encapsular a criação de uma classe e colocar o conhecimento da classe para fora do *framework*, tomando a decisão da forma de instanciar e qual instanciar (GAMMA *et al.*, 2000). O padrão deve ser utilizado quando:

- uma classe não souber qual a classe ou tipo de objetos e suas dependências deve criar;
- a classe especifica que as subclasses tenham conhecimento sobre os objetos que criam; e
- a superclasse deseja delegar uma responsabilidade para uma ou mais subclasses auxiliares e deseja saber o conhecimento da subclasse que foi encarregada.

A figura 5 demonstra uma visualização simplificada do diagrama de classe do *Factory Method*, ilustrando seu funcionamento. A classe *Creator* declara o método *FactoryMethod()* na qual retorna novos objetos. A classe *ConcreteCreator* vai subscrever para retornar uma instância de produto da classe *ConcreteProduct* que são implementações da interface da classe *Product* (define a interface que pode ser produzida pelo *Factory Method*).

Figura 5 – Diagrama de classe do padrão factory method



Fonte – Adaptado de Gamma *et al.* (2000).

Após abordar o propósito criacional, sobre os mecanismos para abstrair a criação de objetos, a seguir, serão apresentados alguns padrões de atuação estrutural, explorando outra categoria dos *design patterns* GoF.

### 2.3.2 Padrões Estruturais

Os padrões estruturais descrevem como relacionar objetos e classes para formação de estruturas maiores. Podem ser aplicadas a classes e objetos, na qual os padrões de estruturas de classes usam herança para definir implementações ou interfaces, enquanto os padrões estruturais de objetos mostram formas de complementar objetos para prover novas funcionalidades. A flexibilidade adicionada da composição de objetos provém da capacidade de mudança da composição em tempo de execução, não sendo possível na composição de classes estáticas (GAMMA *et al.*, 2000). A seguir, serão apresentados os padrões *Proxy* e *Adapter*.

#### 2.3.2.1 Proxy

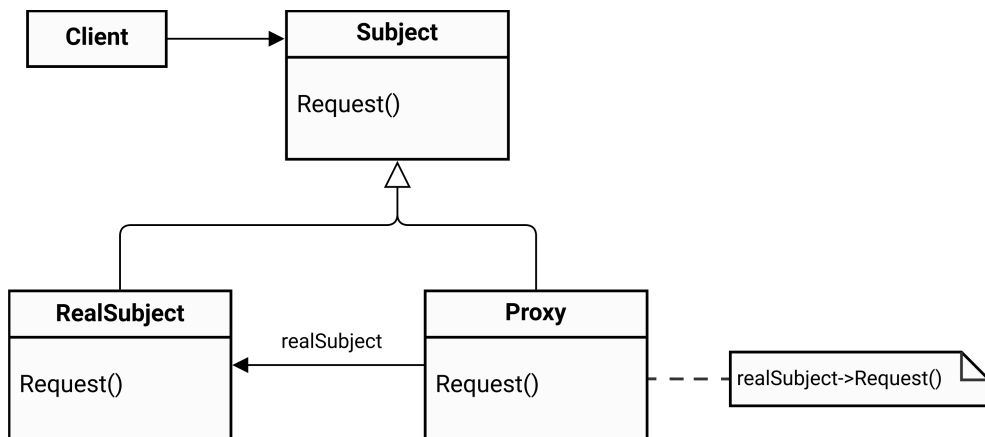
O padrão *proxy* visa fornecer um substituto ou ponto de localização de outro objeto. O segundo objeto, chamado de “proxy” passa a controlar o acesso a esse objeto original (GAMMA *et al.*, 2000).

Conforme Gamma *et al.* (2000), o padrão *Proxy* controla o acesso a um objeto e ele pode ser utilizado, por exemplo, quando um objeto grande que consome muitos recursos para serem criados, torna a inicialização lenta, assim deve-se criar apenas quando realmente for necessário. A solução que o *proxy* sugere é usar um objeto (*Proxy*), substituto temporário do objeto original, que ao receber uma solicitação, é feita a criação do objeto real que as repassam.

A figura 6 demonstra a implementação do padrão *Proxy*, em que a classe *Proxy* mantém uma referência que aponta para o objeto real (representada pela classe *RealSubject*) e ambos fornecem uma interface do serviço (*classe Subject*). A classe *Proxy* passa a representar a classe *RealSubject*, controlando seu acesso e sua criação.

A seguir, será apresentado o padrão *Adapter* que possui outro método de proposta estrutural.

Figura 6 – Diagrama de classe do padrão Proxy



Fonte – Adaptado de Gamma *et al.* (2000).

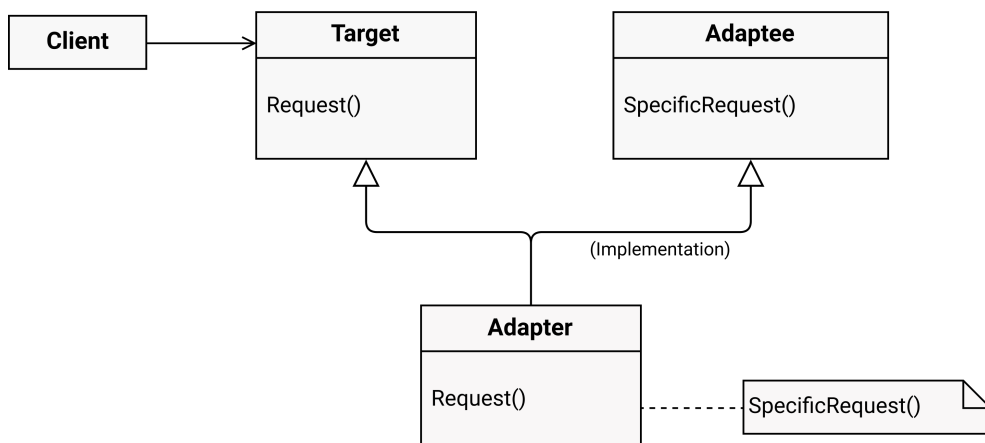
### 2.3.2.2 Adapter

O *design pattern Adapter* permite a colaboração de interfaces incompatíveis entre si. Segundo Gamma *et al.* (2000), a intenção é converter a interface de uma classe em outra interface, esperada pelos clientes. O *Adapter* permite que classes com interfaces incompatíveis trabalhem em conjunto.

O *Adapter* deve ser utilizado quando deseja-se usar uma classe existente, porém sua interface não corresponde a interface necessária, para criar uma classe reutilizável que consiga colaborar com classes não relacionadas ou não previstas.

Para adaptações de classes, conforme figura 7, ocorre a composição do objeto por uma interface implementada pelo Adaptador.

Figura 7 – Diagrama do padrão Adapter - Adaptador de classe

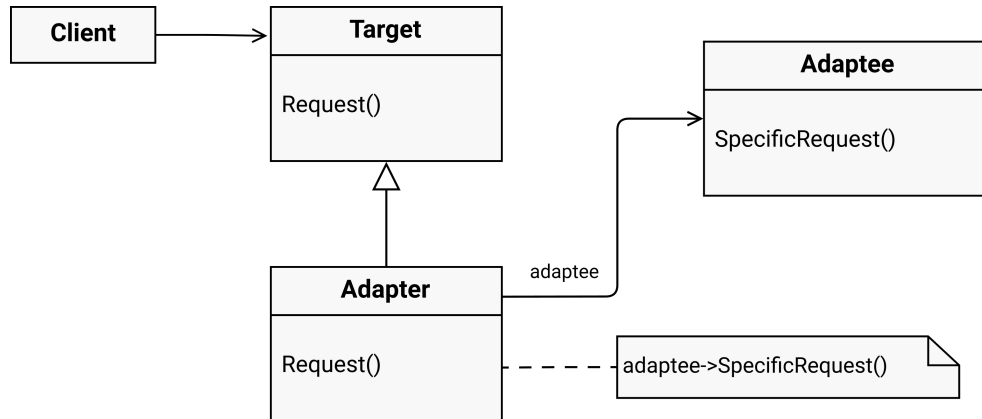


Fonte – Adaptado de Gamma *et al.* (2000).



Para adaptações de objetos, conforme figura 8, a implementação usa a herança e assim a interface de ambos os objetos são herdadas pelo adaptador. Essa abordagem só pode ser utilizada quando a linguagem suporta herança múltipla.

Figura 8 – Diagrama do padrão *Adapter* - Adaptador de objeto



Fonte – Adaptado de Gamma *et al.* (2000).

Apesar das vantagens na utilização do *Adapter* para permitir os ajustes, a complexidade geral do código é aumentada, por adicionar um conjunto de novas classes e interfaces.

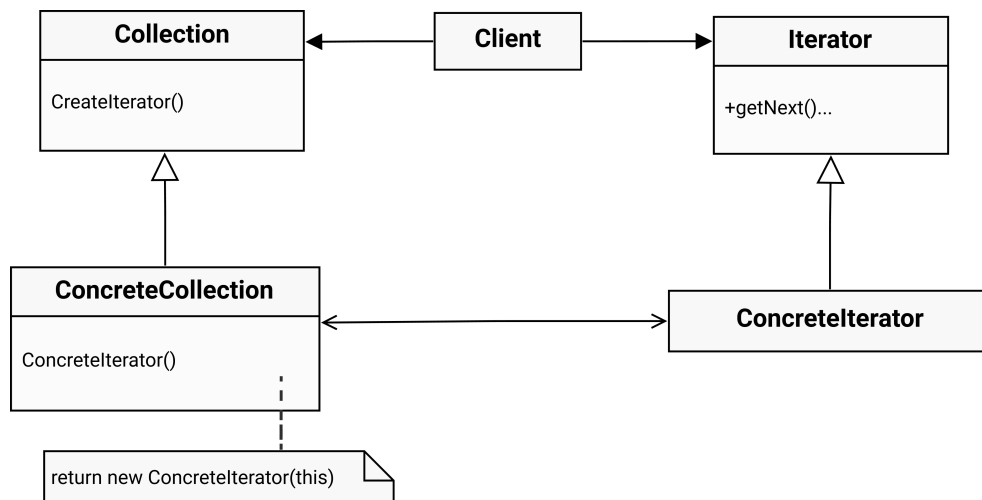
### 2.3.3 Padrões Comportamentais

Os padrões comportamentais lidam com algoritmos e a atribuição de responsabilidades entre objetos, além de descrever padrões de comunicação entre classes e objetos. Os padrões relacionados às classes usam herança para distribuir seus comportamentos, em vez dos padrões comportamentais de objetos, que utilizam composição de objetos (GAMMA *et al.*, 2000). A seguir, serão apresentados os padrões *Iterator* e *State*, pertencentes ao padrão comportamental.

#### 2.3.3.1 Iterator

O padrão *Iterator* possibilita um meio de acesso, percorrendo os elementos de um objeto agregado de forma sequencial sem expor suas representações internas, e deve ser utilizado quando deseja-se acessar o conteúdo do objeto sem expor sua representação interna. O padrão reduz a duplicação de código, além de percorrer diferentes estruturas de objetos agregados, provendo uma interface uniforme (PEREIRA, 2008). A figura 9 apresenta um diagrama de classe mostrando o comportamento da aplicação do padrão *Iterator*.

Figura 9 – Diagrama do padrão Iterator



Fonte – Adaptado de Gamma *et al.* (2000).

Esse padrão possui três consequências principais: permite variações no caminho de objetos agregados; possibilita existir vários percursos ocorrendo ao mesmo tempo em um objeto agregado; e a simplificação da interface do agregado. A seguir, será retratado o padrão *State* que contém outra abordagem de comportamento.

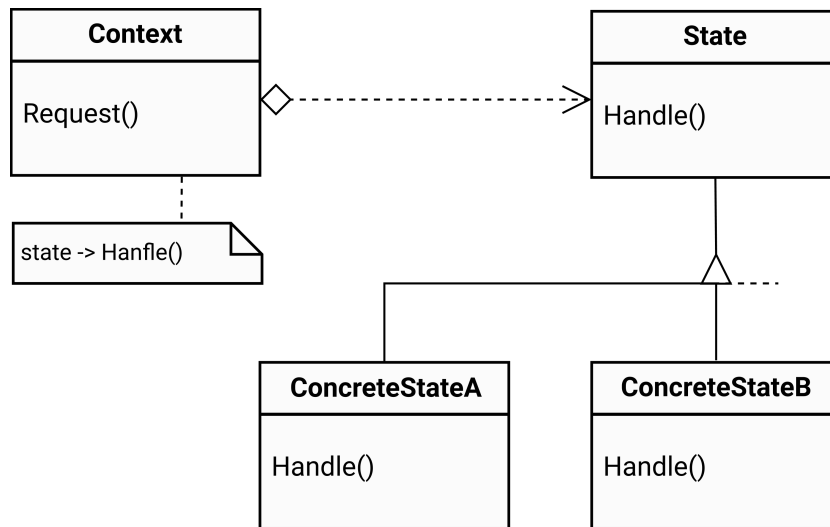
### 2.3.3.2 *State*

O padrão *State* permite que o comportamento de um objeto seja alterado, quando o estado do objeto interno muda, fazendo parecer que o objeto tenha mudado de classe. Gamma *et al.* (2000) afirmam que deve-se utilizar o padrão nas seguintes condições: quando o objeto se comporta de formas diferentes dependendo do estado e o mesmo muda em tempo de execução ou quando a classe contém estruturas condicionais grandes, que se comporta dependendo do estado do objeto.

A estrutura do padrão *State* é ilustrada na figura 10. A classe *Context* armazena uma instância de um objeto concreto (subclasses *ConcreteStateA* e *ConcreteStateB*) que define seu estado atual. O *State* define uma interface para encapsular responsabilidades associadas com um específico *Context*. O *ConcreteState* implementa as responsabilidades ou comportamentos de um *Context*.

A implementação do padrão *State* traz as seguintes consequências: coloca todos os comportamentos específicos de um estado em um objeto. Porém, tal distribuição aumenta o

Figura 10 – Diagrama do padrão State



Fonte – Adaptado de Gamma *et al.* (2000).

número de classes, mas evita o uso de grandes condicionais, se existirem muitos estados. Além de tornar as transações de estado de maneira explícita.

Os *design patterns* mais conhecidos são os padrões GoF, porém existem vários outros padrões catalogados, que serão apresentados na próxima seção.

#### 2.3.4 Arquiteturas e Padrões Web

Segundo Pressman (2006), a arquitetura de software de um programa ou sistema computacional é a estrutura ou estruturas do sistema, que abrange os componentes de software, as propriedades externamente visíveis desses componentes e as relações entre eles. Conforme proposto por Garlan e Shaw (1994), a arquitetura de software é um aspecto do design que vai além dos algoritmos e estruturas de dados, assim envolve: protocolos de comunicação, direcionar o negócio, definir as estruturas que formarão o sistema, atribuir funcionalidades ao sistema, protocolos de comunicação, distribuição física, escalonamento e desempenho e a geração de alternativas no projeto (NING *et al.*, 2008).

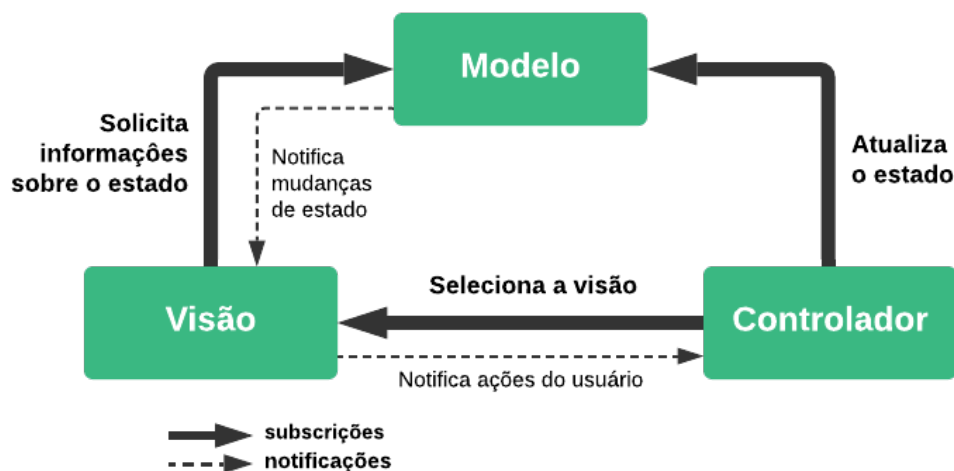
Padrões de arquitetura é uma solução geral aos problemas de arquitetura de software, para uso de recursos ou a infraestrutura. Dentre os diversos padrões de arquitetura para desenvolvimento de sistemas, o presente trabalho destaca o padrão MVC - Model View Control, o padrão Front Controller e o padrão DAO - Data Access Object.

### 2.3.4.1 MVC - Model-View-Controller

O MVC é um padrão de arquitetura de software responsável por desacoplar a interface da navegação e regras de negócios, permitindo mais flexibilidade e maior reutilização. Esse padrão otimiza a velocidade entre as requisições da interface do usuário e dados no banco. De acordo com Duarte (2011), essa abordagem auxilia na criação de diferentes interfaces com o usuário para um mesmo modelo sem a necessidade de alteração, sendo um aspecto importante em aplicações Web, que em sua maioria, são empregadas em diferentes plataformas, que podem receber tratamentos de interface distintos, preservando o mesmo modelo em um único servidor remoto ou em bases distribuídas.

A arquitetura MVC auxilia na construção de aplicações separando o projeto em três camadas principais bem definidas, ilustrada na figura 11, sendo elas, o *Model*, o *Controller* e a *View*, cada uma executando somente o que lhe é definido<sup>1</sup>.

Figura 11 – Relacionamento entre as camadas da arquitetura MVC



Fonte – <http://java.sun.com/developer/technicalArticles/javase/mvc/index.html>

Cada uma das camadas são apresentadas a seguir:

- **Modelo (Model):** engloba a parte lógica da aplicação que gerencia o comportamento dos dados, por meio das funções, lógica e regras de negócios. Responsável pela leitura, escrita e validação;
- **Visão (View):** camada de interação com o usuário, permite a apresentação do conteúdo e da lógica, mostrando as informações e funcionalidades de processamento

<sup>1</sup> <<http://java.sun.com/developer/technicalArticles/javase/mvc/index.html>>

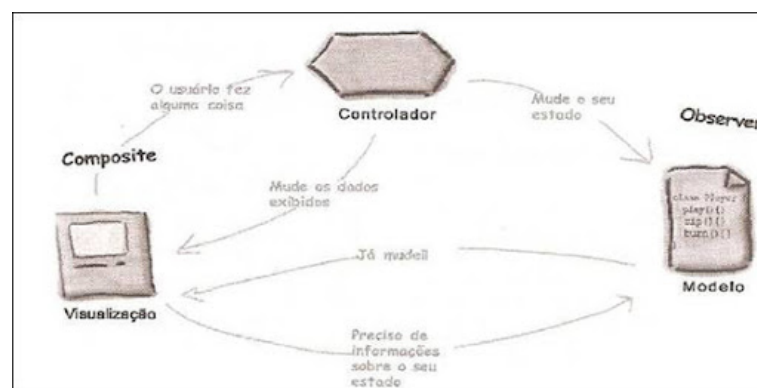
requeridas pelo usuário; e

- **Controlador (*Controller*):** responsável pela comunicação entre interfaces e coordenar a regras de negócios, controlando o acesso ao *model* e qual *view* será mostrado.

A comunicação entre as camadas na arquitetura MVC, ocorre de forma triangular: a visão dispara uma modificação para o controlador (exemplo: Preenchimento de um formulário), após capturar esse sinal, solicita uma ação para o Modelo. Assim, o modelo tem seu estado atualizado, o Controlador informa a visão sobre o novo estado, a visão busca os dados diretamente do Modelo e exibe o resultado para o usuário (LEMOS *et al.*, 2013).

A arquitetura MVC contém um conjunto de *design patterns* trabalhando em suas camadas. De acordo com Gamma *et al.* (2000), os principais padrões que o MVC utiliza são os Observer, Composite e o Strategy. Abordando os padrões comportamentais Observer e Strategy e o padrão estrutural Composite. Conforme figura 12, o View juntamente com o padrão Composite fica responsável pela espera de eventos solicitados pelos usuários. O Model por sua vez aplica o Observer para ser independente das visualizações e controladores. Já o Controller adota o Strategy para manter a View separado das decisões sobre o comportamento da interface.<sup>2</sup>

Figura 12 – *Design patterns* na arquitetura MVC.



Fonte – Freeman *et al.* (2009)

#### 2.3.4.2 *Front Controller*

O *design pattern Front Controller* se comporta como um mecanismo centralizado para tratamento de solicitações, em que todas as solicitações são tratadas por um único contro-

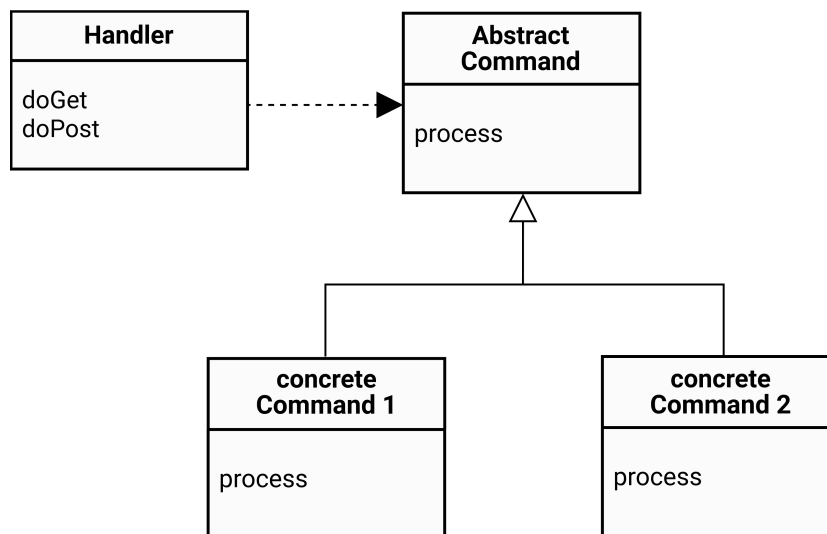
<sup>2</sup> <<http://www.linhadecodigo.com.br/artigo/2367/abordando-a-arquitetura-mvc-e-design-patterns-observer-composite-strategy.aspx>>

lador. Esse manipulador pode fazer a autenticação, autorização, registros ou rastreamento da solicitação, para então direcionar ao manipulador adequado.

Sites complexos possuem muitas funcionalidades semelhantes que o desenvolvedor precisa realizar ao tratar solicitações como: segurança, internalização e fornecimento de apresentações em resposta para determinados usuários. Se o comportamento do controlador de entrada for espalhado para diversos objetos, é possível ter a duplicação deste comportamento, além de ser difícil alterar o comportamento em tempo de execução (FOWLER, 2002).

O Front Controller é estruturado em duas partes: um manipulador Web (*Handler*) e uma hierarquia de Comandos (*Command*). A figura 13 ilustra o Front Controller. O manipulador Web recebe todas as solicitações HTTP do tipo POST ou GET do servidor Web, analisando as informações da URL para decidir que tipo de comando executar.

Figura 13 – Diagrama de classes do padrão arquitetural *Front Controller*



Fonte – Adaptado de FOWLER (2002)

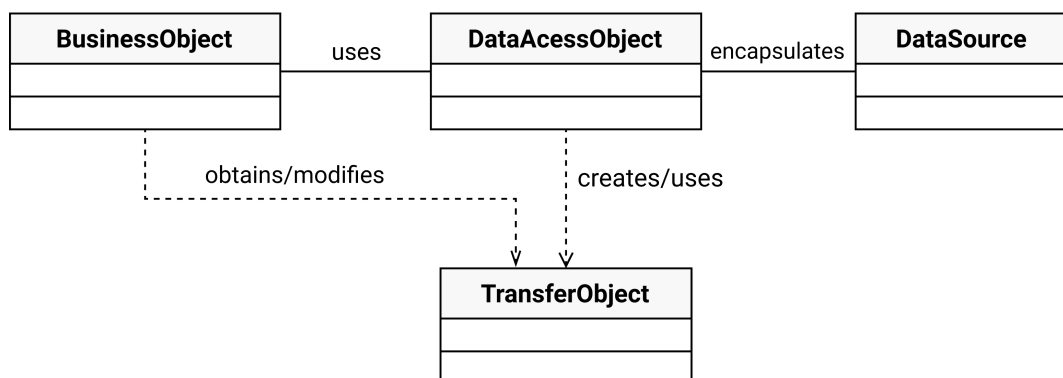
O manipulador Web é implementado como uma classe ao invés de uma página servidora, já que não produz nenhuma resposta. Os comandos também são classes em vez de páginas servidoras. O manipulador Web é normalmente algo simples que somente decide qual comando executar. O *Front Controller* é uma parte fundamental para o uso do padrão MVC, ficando responsável por toda entrada e roteamento dos pedidos da aplicação, sendo tratados em um único local (SCUR, 2012).

### 2.3.4.3 DAO - Data Access Object

O padrão DAO consiste em encapsular os mecanismos de acesso e persistência dos dados da lógica da aplicação. O DAO cria uma interface genérica para a camada de negócios para seus clientes, abstraindo as particularidades da execução da origem dos dados. A interface exposta pelo DAO não fica dependente da implementação da fonte de dados utilizada. Esse padrão permite criar as classes de dados se adaptando a diferentes esquemas de armazenamento sem afetar os clientes ou a camada de negócios (SRIDARAN *et al.*, 2009).

O DAO surgiu com a necessidade de separar as regras de negócio das regras de acesso a banco de dados. Por ser um padrão voltado ao paradigma orientado a objetos, o problema de incompatibilidade entre lógica da orientação a objetos e a lógica relacional é resolvido parcialmente (VIDAL, 2016).

Figura 14 – Diagrama de classe mostrando os relacionamentos para o padrão DAO



Fonte – <https://www.oracle.com/java/technologies/dataaccessobject.html>

De acordo com Sardagna e Vahldick (2008), o relacionamento no DAO ocorre da seguinte forma:

1. **DataAccessObject:** a classe principal desse padrão, encapsula a implementação de acesso aos dados, contém os mapeamentos para o *BusinessObject* obter acesso a fonte de dados, que opera carregamento e armazenamento de dados para ao *DataAccessObject*;
2. **DataSource:** mantém a fonte de dados que podem ser um BD, XML ou seja, a origem dos dados;
3. **BusinessObject:** representa o cliente do padrão, contém a lógica de negócio e usa o objeto *DataAccessObject*;
4. **TransferObject:** esse objeto contém os dados que transitam a fonte de dados.

Os *design patterns* serão utilizados nesse trabalho para verificar a viabilidade e o desempenho quando utilizados em aplicações Web que usam *frameworks Web JavaScript*. A próxima seção apresenta os *frameworks* que serão utilizados.

## 2.4 Frameworks Web JavaScript

*Framework* é uma estrutura para a construção de aplicações e dentre outros projetos digitais, que traz soluções prontas aos desenvolvedores, trazendo um conjunto de códigos pré-definidos que auxiliam o desenvolvimento de novos projetos.

De acordo com Duarte (2015), *Frameworks* são constituídos de padrões, aplicando um *template* que os desenvolvedores devem seguir para criar projetos, objetivando obter todo o poder do *framework*, e que contém uma maneira de desenvolvimento específico e espaços vazios que serão preenchidos pelos desenvolvedores, a fim de trabalhar o código para as necessidades do projeto.

Os *frameworks* já fazem parte do dia a dia dos programadores, principalmente nos projetos com grandes números de funções similares. A praticidade em iniciar projetos com *frameworks* apresenta diversas vantagens no desenvolvimento de aplicações. De acordo com Fayad *et al.* (1999), sua utilização traz os seguintes benefícios:

- **Melhora a modularização** – A implementação dos detalhes voláteis são encapsulados em interfaces estáveis, onde o usuário não consegue modificá-lo;
- **Aumenta a reutilização** – Desenvolver componentes genéricos para reaplicar na produção de novos sistemas futuros;
- **Extensibilidade** – O uso dos métodos *hooks* permitem que as aplicações compreendam interfaces estáveis; e
- **Inversão de controle** – O *framework* controla o fluxo global de execução dos programas em vez de ser o desenvolvedor.

No entanto, é preciso investir na qualidade do projeto do *framework*. O processo de construção com *frameworks* acaba sendo mais complexo do que em aplicações tradicionais. Deve ser planejado para obter realmente o reuso prometido, modificando seu processo de desenvolvimentos e aplicando novos incentivos.



Devido ao grande número de *frameworks JavaScripts* para desenvolvimento de aplicações Web, foram selecionados os mais populares para a abordagem dessa pesquisa: Angular, React e Vue.js.

A seleção teve como base avaliativa referente a popularidade da comunidade, a coleta e análise de grandes sites especializados em controle de versão e plataformas relacionadas ao desenvolvimento de software, sendo o *GitHub* e *Stack Overflow* como fontes de dados. A Tabela 1 apresenta os dados coletados do Github.

Tabela 1 – Dados dos *Frameworks* no *GitHub*

	Angular	React	Vue.js
# Watchers	3.2k	6.7k	6.3k
# Stars	70.9k	164k	200.8k
# Forks	18.6k	32.9k	31.7k
# Contributors	1,352	1,533	382

Fonte – GitHub

Com relação as estatísticas do *GitHub*, foram analisados os seguintes pontos: observadores, estrelas, cópias do código original do *framework* (*forks*) e contribuidores, conforme a Tabela 1. De acordo com a tabela, é possível observar que o React tem o maior números de observadores, contribuidores e forks. No entanto, o número de estrelas do Vue.js demonstram sua popularidade entre os usuários.

O *Stack Overflow*, como ilustrado na figura 15, utiliza a ferramenta *Stack Overflow Trends* da plataforma, para obter o comparativo do número de questões registradas por *framework* Web.

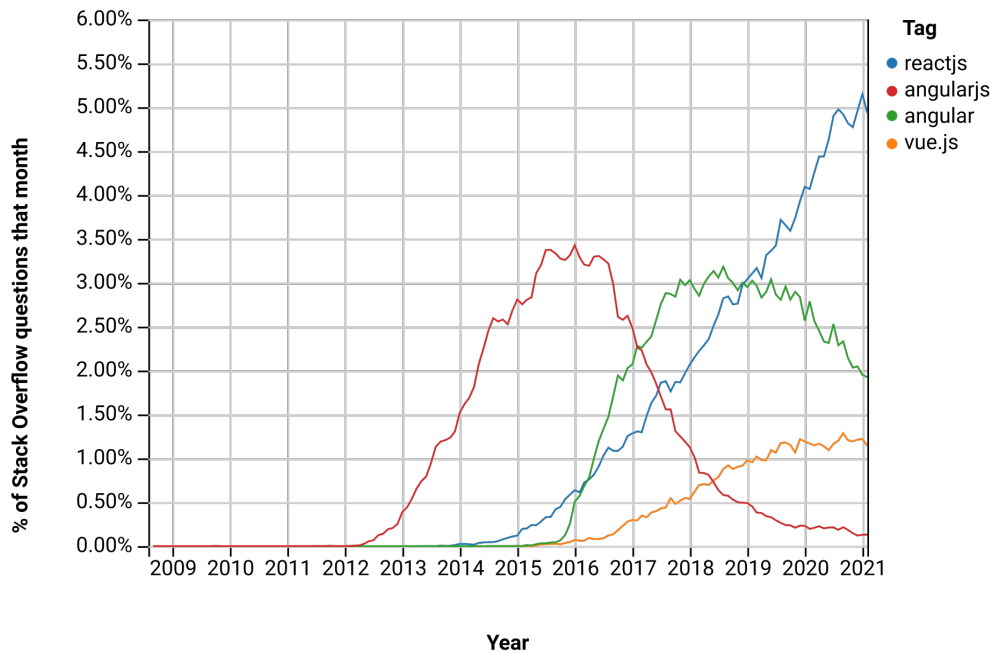
#### 2.4.1 React

Conforme definido em no *slogan* oficial, React é uma biblioteca *JavaScript* declarativa, eficiente e flexível para a construção de interfaces de usuário<sup>3</sup>. React não é um *framework*, como mencionado na seção acima, ele é uma coleção de funções úteis que podem ser usadas pelo desenvolvedor para resolver um determinado problema.

O React adota a sintaxe JavaScript Xml, uma extensão de sintaxe para *JavaScript*.

<sup>3</sup> <<https://reactjs.org/>>

Figura 15 – *Frameworks Web* com o maior numero de questões no Stack Overflow



Fonte – <https://insights.stackoverflow.com/trends?tags=r%2Cstatistics>

Contém *tags* que descrevem e compreende como será montado a UI. O JSX possibilita colocar estruturas HTML em *JavaScript*. Não é necessário usar JSX no React, mas sua utilização ajuda a escrever as aplicações, ao disponibilizar mensagens mais úteis de erro e aviso<sup>4</sup>.

Em comparação aos elementos DOM do navegador, os elementos React são objetos simples e menos custosos em recursos na criação. O Virtual Dom do React é responsável por buscar mudanças em cada componente, como demonstrado na figura 16. Ele aplica as modificações somente nos componentes que mudaram de estado<sup>5</sup>.

Os componentes React permitem que a UI seja dividida em partes independentes e reutilizáveis, tendo cada parte pensada isoladamente. Os componentes são como funções JavaScript que aceitam entradas (chamadas de props) e retornam elementos React, retratando o que vai ser mostrado na tela.

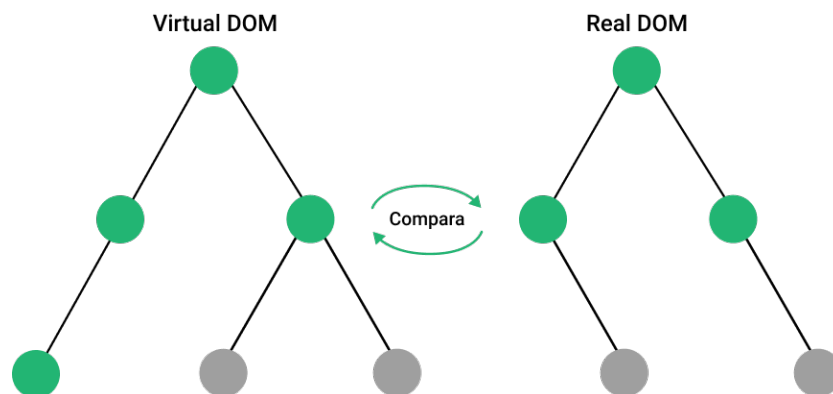
## 2.4.2 *Vue.js*

O Vue.js (conhecido como Vue) é um *framework* progressivo de código aberto, versátil e eficiente na construção de interfaces de usuário, projetado para ser adotado de forma

<sup>4</sup> <<https://reactjs.org/docs/introducing-jsx.html>>

<sup>5</sup> <<https://reactjs.org/docs/rendering-elements.html>>

Figura 16 – Árvore de componentes do React, compara as diferenças entre o DOM virtual e o DOM real

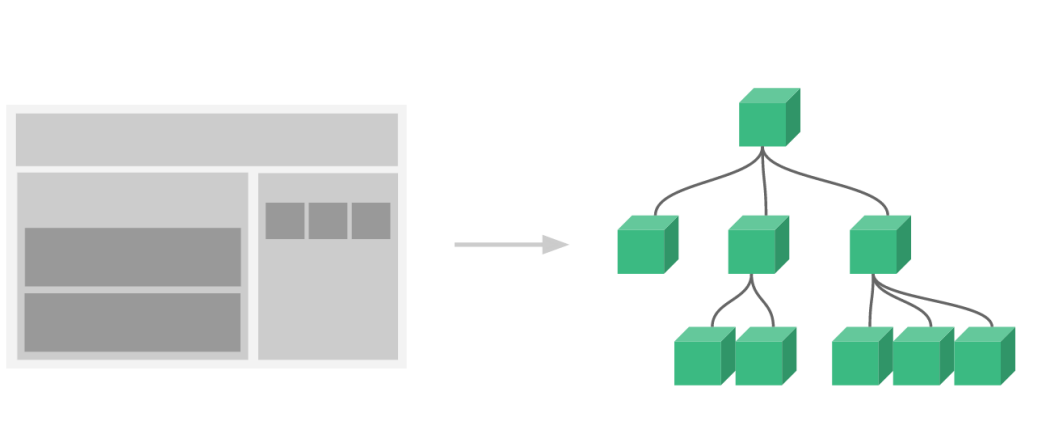


Fonte – Adaptado de Saks (2019).

incremental<sup>6</sup>. O Vue.js é uma estrutura *JavaScript*, criada por Evan You, ex-funcionário do *Google* que após trabalhar com Angular, sentiu que utilizá-lo em certos casos era muito pesado, e que poderia extrair partes que gostava no Angular para construir um *framework* leve.

O Vue.js é constituído de componentes reutilizáveis que encapsula partes da aplicação, a fim de permitir construir aplicações maiores com pequenos componentes independentes e altamente reutilizáveis. Aplicativos frequentemente tem sua interface organizada em uma árvore de componentes, ilustrada na figura 17. Os componentes podem ser registrados de forma global, acessado por todos os subcomponentes ou de forma local. O Vue.js baseia sua sintaxe em HTML, CSS e *Javascript* em uma única instância Vue<sup>7</sup>.

Figura 17 – Interface visual da aplicação, abstraída por uma árvore de componentes



Fonte – Filipova (2016)

<sup>6</sup> <<https://vuejs.org/v2/guide/>>

<sup>7</sup> <<https://vuejs.org/v2/guide/syntax>>

O *framework* possui uma renderização semelhante ao React, baseada em um virtual DOM para atualizar a página sempre que houver mudança de dados nos componentes, o qual aumenta a performance diminuindo a quantidade de atualizações desnecessárias. Cada instante do Vue.js deve ser configurado para observar as mudanças.

A aplicação do Vue.js possui uma curva de aprendizado menor, se comparado aos outros *frameworks* Angular e React. Ele oferece uma capacidade de personalização, o qual permite que o desenvolvedor estruture o aplicativo de forma flexível. No entanto, o código pode se tornar difícil de depurar e testar.

### 2.4.3 Angular

O Angular é um *framework JavaScript* para desenvolvimento de aplicativos, focado na construção de interfaces de usuário e integra servidores que fornecem interface em REST/JSON. Com esse *framework* é possível construir aplicações Web de página única de forma eficiente e otimizada, para diferentes dimensões, sejam elas para a Web, *mobile* ou *desktop*, tornando-as mais escaláveis e dinâmicas<sup>8</sup>.

O Angular começou em 2009 e foi desenvolvido por Misko Hevery e Adam Abrons. Ambos decidiram distribuir o Angular como uma plataforma *open-source*, mas Hevery, que trabalha na *Google*, manteve a plataforma com alguns colegas da empresa: Igor Minár e Vojta Jína. Em 2016 foi totalmente reescrito e tornou-se uma estrutura completamente nova. A primeira versão passou a ser chamada de Angular.js e versões superiores denominadas de Angular.

A plataforma de desenvolvimento Angular passou a ser inteiramente baseada no *TypeScript*, um superconjunto *JavaScript* que adiciona tipagem estática ao código<sup>9</sup>. A sintaxe do *template* Angular é HTML, em que quase toda a sintaxe HTML é válida, exceto a tag `<script>` para eliminar o risco de ataques injeção de *script*<sup>10</sup>.

A arquitetura de um projeto Angular é desenvolvido utilizando componentes em árvore. Os componentes da aplicação devem ter uma comunicação entre si, sendo organizados em *NgModules* que fornecem um contexto para os componentes serem compilados para um

---

<sup>8</sup> <<https://angular.io/docs>>

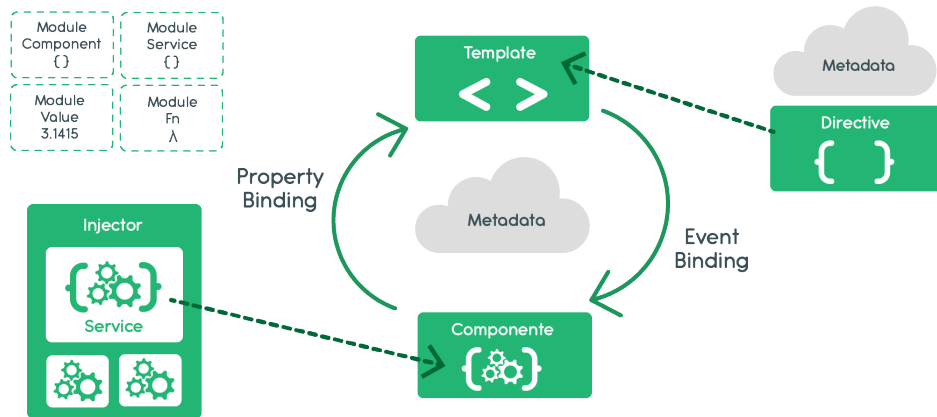
<sup>9</sup> <<https://angular.io/guide/what-is-angular>>

<sup>10</sup> <<https://angular.io/guide/template-syntax>>

determinado domínio, existindo pelo menos um módulo raiz<sup>11</sup>. Os elementos básicos do Angular são:

1. **Componente:** define uma classe, que está associada a um *template* HTML e os serviços para atribuir uma funcionalidade para seu módulo;
2. **Service:** contém toda a regra de negócio da aplicação;
3. **Diretivas:** adiciona as classes, definindo comportamentos específicos nos elementos DOM;
4. **Módulo:** agrupa componentes, serviços e diretivas a um objetivo ou funcionalidade.

Figura 18 – Os elementos básicos de uma aplicação Angular



Fonte – Afonso (2018)

## 2.5 Considerações Finais

Este capítulo apresentou os conceitos básicos sobre os temas do trabalho proposto: Engenharia de Software e processos de desenvolvimento, a evolução dos processos de desenvolvimento de Software para Web, *Design Patterns* e as arquiteturas, os *frameworks front-end* Web utilizados na pesquisa. Com relação aos temas foram apresentadas definições e conceitos básico para o entendimento do trabalho, assim como os *design patterns* e *frameworks front-end* Web que serão utilizados. Existem vários trabalhos que fazem o uso dessa abordagem, dois serão discutidos no próximo capítulo, a fim de verificar seus parâmetros e resultados comparativos.

<sup>11</sup> <<https://angular.io/guide/ngmodules>>

### 3 TRABALHOS RELACIONADOS

Este capítulo tem por objetivo apresentar e comparar pesquisas relacionadas com o tema abordado no presente trabalho, que adotam desenvolvimento Web com aplicação de *design pattern* e *frameworks*.

#### 3.1 Trabalho de Cechinel *et al.*

De acordo com Cechinel *et al.* (2017), o desenvolvimento Web é constituído pelo processo de construção e aplicação de testes específicos para a Web, o qual foi identificado alguns problemas presentes na área de desenvolvimento Web, sendo aspectos críticos como segurança, usabilidade e a manutenabilidade.

O trabalho de Cechinel *et al.* (2017), consiste em abordar e analisar três *frameworks* ou bibliotecas usadas no desenvolvimento de aplicações Web atualmente e compará-los com JavaScript puro. Para alcançar esses resultados, os autores propõe a aplicação de exemplo de igual aparência aos *frameworks* ou bibliotecas, a fim de explorar as diferenças, avaliando usabilidade, custo, eficiência e funcionalidade.

No decorrer do trabalho é realizado um estudo sobre tecnologias fundamentais para o desenvolvimento Web e os *design patterns* JavaScript utilizados nos *frameworks* e bibliotecas. Foi definida a escolha de um *framework* (AngularJS) e duas bibliotecas (React e Knockout) para o desenvolvimento da aplicação de exemplo, além do JavaScript puro.

O trabalho realiza uma avaliação e análise de desempenho, tendo a eficiência (desempenho e tamanho) como o principal fator para a escolha de um *framework* JavaScript. O tempo de execução foi medido em três diferentes navegadores, verificando o tamanho do *framework* e bibliotecas, quantidade de código JS e quantidade de código HTML. Por fim, apresenta uma comparação das diferentes características do Angular, React e Knockout<sup>1</sup>. Realizou-se um estudo sobre o estilo arquitetural adotado por cada *framework* e biblioteca aplicada, onde o Angular adota uma estrutura semelhante ao MVC, o Knockout segue o padrão Model-View-ViewModel (MVVM) e o React aplica a arquitetura Flux. O Angular apresentou uma estrutura mais complexa, se comparado ao React e Knockout. Já no desempenho, o React demonstra ter

---

<sup>1</sup> <<https://knockoutjs.com/>>

uma melhor estrutura de aplicações Web. No entanto, o Angular se mostrou mais ágil em tempo de desenvolvimento.

O trabalho aqui proposto também se concentra em análise de *frameworks* e bibliotecas, mas diferente do que foi proposto por Cechinel *et al.* (2017), é executado uma pesquisa sobre os *design patterns* utilizados no desenvolvimento de software, que além de comparar a nível de desempenho, faz uma comparação de métricas de software com e sem *design pattern*.

### 3.2 Trabalho de Mantoan

Mantoan (2009) apresenta um estudo sobre o desenvolvimento e a implementação de um software com a utilização de *design patterns*, especificamente os padrões GoF, para construir uma arquitetura de desenvolvimento de uma aplicação Web de uma forma mais padronizada, facilitando a manutenção, menos repetição de código e evitando soluções simples sejam implementada utilizando muitas linhas de código desnecessárias.

Para resolução desses problemas, o autor propôs uma arquitetura em PHP para desenvolvimento de aplicações, contendo *design patterns* para fornecer uma padronização de programação e uma maior estrutura organizacional. Para a escolha dos *design patterns* foi necessário um estudo sobre os diversos encontrados, os conceitos de arquiteturas, além do PHP e seus *frameworks*.

O trabalho implementa uma aplicação de gestão de bibliotecas, em que foi definido o Zend Framework<sup>2</sup> para o desenvolvimento da arquitetura e da aplicação. Na implementação foram aplicados diversos *patterns* como: *Singleton*, *Factory*, *Observer*, *Data Mapper* e *MVC*, com objetivo de prover uma estrutura de software compatível com diversos tipos de aplicações. A análise comparativa proposta nesse trabalho, tem foco em analisar os benefícios de aplicações Web contendo *design patterns*. Na qual, a implementação utilizada na arquitetura de software desenvolvida se mostrou mais organizada, possibilitando uma maior reusabilidade e uma manutenção muito mais viável, possibilitando que sistemas tenham seu ciclo de vida bastante prolongado ao implementar arquiteturas de software e *design patterns*. Como deficiência desse trabalho, não foi realizado uma análise de desempenho quando é aplicado *design pattern*.

---

<sup>2</sup> <<https://framework.zend.com/>>

### 3.3 Trabalho de Thung *et al*

De acordo com Thung *et al.* (2010), a adoção de *design patterns* no desenvolvimento de aplicações Web estabelecem soluções mais eficientes, resultando em sua perspectiva de produção, um projeto mais reutilizável e sustentável. Enquanto da perspectiva dos usuários, tais soluções fornecidas pelos *design patterns* estabelecem melhorias de usabilidade, na qual ajudam a encontrar soluções para problemas específicos no projeto, onde sua não adoção no processo de desenvolvimento consumiria tempo e dificultaria manutenções futuras.

Thung *et al.* (2010) abordam um estudo sobre viabilidade de *design patterns* com foco em design arquitetônico e padrões de navegação para construção de aplicativos Web. Eles analisam e comparam seus impactos, a fim de estabelecer os *Patterns* mais adequados no contexto de aplicações Web.

Para resolução da pesquisa é realizado um estudo de caso no site da escola, a fim de compreender melhor a aplicação dos *design patterns* no desenvolvimento Web, contendo *design patterns* para fornecer uma padronização de estrutura e navegação do site. Foi definida a escolha dos padrões arquitetônicos Model-View-Controller (MVC) e Presentation-Abstraction-Control (PAC) e cinco padrões de navegação que são observador de navegação, estratégia de navegação, notícias, paginação e navegação baseada em conjunto.

O trabalho realiza uma análise baseada na estrutura, contexto e suas consequências para cada design patterns proposto, realizando uma investigação do projeto original antes da sua adoção, na qual, a implementação utilizada na arquitetura do site melhorou sua usabilidade e tornou sua manutenção para designers e desenvolvedores de software mais eficiente e escalável. No entanto, o trabalho aqui proposto tem foco em analisar o desempenho de aplicações Web desenvolvidas com *frameworks* JavaScript contendo design patterns, diferente da comparativa realizada por Thung *et al.* (2010), que apresenta as consequências em usar cada *design pattern* em aplicativos Web.

### 3.4 Comparativo Entre os Trabalhos

Para o comparativo entre os trabalhos relacionados, foram levantadas características relevantes desses trabalhos em relação a pesquisa proposta. As características consideradas para



a análise são as seguintes:

- **Métricas de desempenho:** essa característica diz respeito a informar se a solução analisou métricas de desempenho na comparação.
- **Métricas de software:** refere-se a informar se a solução analisou métricas de software na comparação.
- **Comparação Antes / Depois:** essa característica refere-se se a solução realizou a comparação antes e depois de aplicado os padrões.
- **Quantidade de Padrões:** essa característica diz respeito a quantidade de padrões que foram utilizadas na solução.

A Tabela 2 detalha o comparativo entre os trabalhos relacionados e as características elencadas.

Tabela 2 – Comparativo Entre os Trabalhos Relacionados

Trabalho	Métricas de Desempenho	Métricas de Software	Comparação Antes / Depois	Quantidade de Padrões
Cechinel et al.	Sim	Sim	Sim	3
Mantoan	Sim	Não	Sim	5
Thung et al	Sim	Não	Sim	7

Fonte – Autoria própria

### 3.5 Considerações Finais

Esse capítulo apresentou os trabalhos relacionados, que envolvem desenvolvimento Web com aplicação de *design pattern* e *frameworks*. Foi verificado que alguns dos trabalhos relacionados não apresentam uma análise do impacto causado do emprego dos padrões em aplicações Web. Assim, o próximo capítulo será apresentada a metodologia adotada no trabalho proposta, que visa apresentar essa análise.

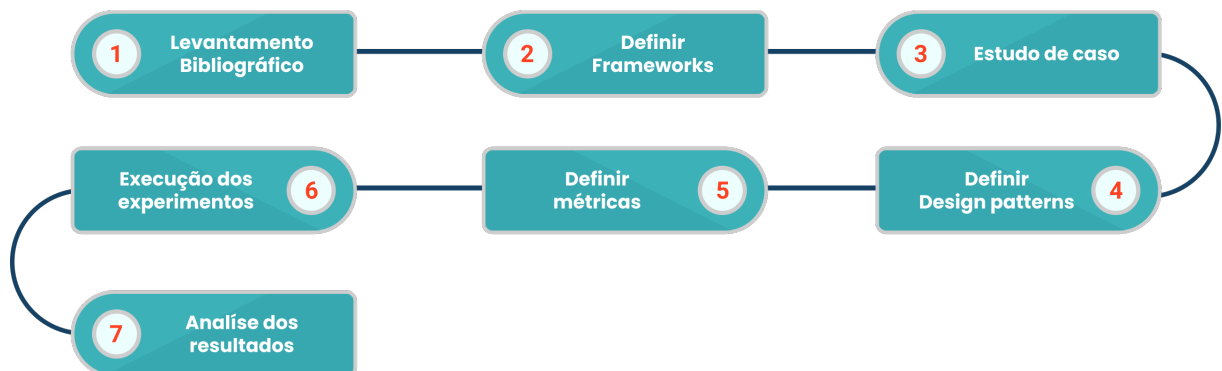
## 4 METODOLOGIA

Neste capítulo será apresentado a metodologia deste trabalho, que é a de realizar uma comparação entre *frameworks front-end* quando aplicado padrões de projeto em aplicações Web.

O trabalho trata-se de um projeto de pesquisa aplicada, visando a realização de um estudo com o objetivo de obter os resultados acerca da problemática apresentada, através de um estudo experimental que consiste em um conjunto de variáveis selecionadas, definição da medida de controle e observação dos dados obtidos nessas variáveis.

O estudo será essencialmente quantitativo, visando a análise e interpretação das informações coletadas nos experimentos executados, compreendendo a viabilidade da aplicação dos *design patterns* nos *frameworks* selecionados (React e Vue.js). A abordagem da pesquisa quanto aos objetivos, caracteriza-se como um estudo exploratório tendo o levantamento bibliográfico sobre os campos de interesse da pesquisa, proporcionando uma maior familiaridade com o problema (GIL, 2002). A figura 19 ilustra os processos metodológicos da pesquisa, apresentando as etapas adotadas.

Figura 19 – Procedimentos Metodológicos



Fonte – Autoria própria

Primeiramente, foi executado os procedimentos de pesquisa, um levantamento bibliográfico prévio, para realizar uma investigação sobre o campo de interesse da pesquisa, a fim de compreender todos os conceitos e elementos fundamentais que relacionam a engenharia de software, desenvolvimento Web, *design patterns* e *frameworks* JavaScript, conforme é definido por Marconi e Lakatos (2003), como importante para um trabalho de pesquisa.

Na segunda etapa são estabelecidos critérios para a escolha dos *frameworks* Web, em que tem o objetivo de verificar na comunidade o que é levado em consideração na escolha deles.

Será desenvolvida uma prova de conceito na terceira etapa. A prova de conceito consiste em um sistema capaz de gerenciar os dados, onde que seja possível utilizar os *frameworks* e *design patterns*.

A quarta etapa se preocupa em selecionar os *design patterns* para implementar na prova de conceito, assim, tendo uma solução mais reutilizável, eficiente e com boa manutenibilidade.

A quinta etapa consistirá na definição de quais métricas de software e métricas de desempenho serão realizadas nos testes da prova de conceito.

A sexta etapa consistirá na realização dos experimentos nos *frameworks* escolhidos, através de duas metodologias principais: a análise da comparação entre a implementação da prova de conceito com e sem *design pattern* selecionados, em cada *framework* escolhido a nível de métricas de software e a nível de desempenho.

A última etapa consistirá na análise comparativa entre os resultados encontrados na etapa anterior.

#### **4.1 Considerações Finais**

Esse capítulo apresentou a metodologia adotada no trabalho proposto, exibindo cada etapa adotada. O próximo capítulo será apresentada a proposta referente a presente pesquisa, que visa realizar uma comparação em relação as padrões quando aplicado a *frameworks* Web.

## 5 PROPOSTA

Neste capítulo será apresentado a proposta deste trabalho, que é a de realizar uma comparação entre *frameworks front-end* quando aplicado padrões de projeto em aplicações Web e que segue a metodologia apresentada no capítulo anterior. A seção 5.1 apresenta os critérios de escolha dos *frameworks* selecionados. A seção 5.2 exhibe a prova de conceito, bem como funcionalidades e diagramas e *screenshots* da aplicação. A seção 5.3 apresenta os *design patterns* selecionados. A seção 5.4 apresenta as métricas de desempenho e software que serão usadas na análise comparativa.

### 5.1 Frameworks Front-End Web

Devido ao grande número de *frameworks JavaScripts* para desenvolvimento de aplicações Web, devem existir critérios para a escolha dos mesmos. Na seção 2.4, foram apresentados três *frameworks*, em que a seleção teve como critérios o alcance da comunidade; facilidade de uso; capacidade dos *frameworks* de desenvolver com *design pattern*.

Dos três *frameworks* apresentados, o trabalho proposto aborda dois deles: React e Vue.js. O React foi escolhido pois além dos critérios apresentados anteriormente, o autor tem uma maior familiaridade, pois trabalha atualmente com a mesma. O Vue.js foi escolhido pois o autor estudou essa tecnologia na disciplina de desenvolvimento de software para Web.

### 5.2 Prova de Conceito

A prova de conceito consiste em um sistema de controle de atividades, destinado a gestão de compromissos dos usuários. Modelada como uma aplicação simples com as quatro operações básicas do CRUD (Create, Read, Update e Delete), replicada nos *frameworks* React e Vue.js, com e sem *design patterns*. Além disso, fará o uso de recursos de bibliotecas de componentes CSS, chamada Bootstrap<sup>1</sup>. Para o desenvolvimento back-end da aplicação será empregada um serviço de serverless, na qual, foi optado a ferramenta Firebase<sup>2</sup> por simplificar tempo e recurso, visando ter seu principal foco no front-end.

---

<sup>1</sup> <<https://getbootstrap.com/>>

<sup>2</sup> <<https://firebase.google.com/>>

O objetivo do sistema de controle de atividades é acompanhar e planejar compromissos, por meio de cartões que representam tais atividades. A seguir, serão definidos os requisitos funcionais (RF) e não funcionais (RNF) da aplicação.

Tabela 3 – Descrição de Requisitos Funcionais

ID	Requisitos	Descrição
RF01	Adicionar Tarefa	O sistema deve permitir o cadastro de tarefas, com suas respectivas informações gerais.
RF02	Buscar Tarefa	Efetuar uma busca no banco do sistema pelo id
RF03	Editar Tarefa	Atualizar as informações de tarefa no sistema pelo id
RF04	Remover Tarefa	Remover tarefas no sistema pelo id
RF05	Listagem de tarefas	O sistema disponibiliza uma lista para visualização de todas as tarefas cadastrados no sistema

Fonte – Autoria própria

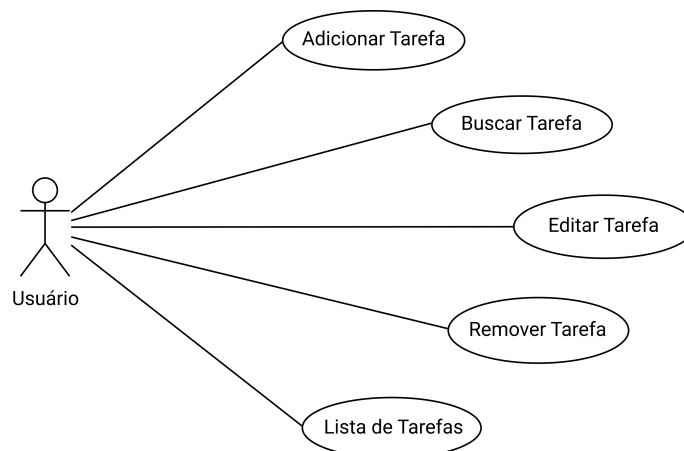
Tabela 4 – Descrição de Requisitos Não Funcionais

ID	Requisitos	Descrição
RNF01	Desenvolvimento	O sistema deve ser desenvolvido para acesso Web utilizando React e Vue.js.
RNF02	Compatibilidade	O sistema deve ser compatível com navegadores Web.

Fonte – Autoria própria

A seguir, é apresentado um diagrama de caso de uso relacionado ao gerenciamento das tarefas da aplicação, que demonstra cada funcionalidade que o usuário conseguirá realizar. A figura 20 contém a modelagem do caso de uso da aplicação desenvolvida.

Figura 20 – Diagrama de Casos de Uso

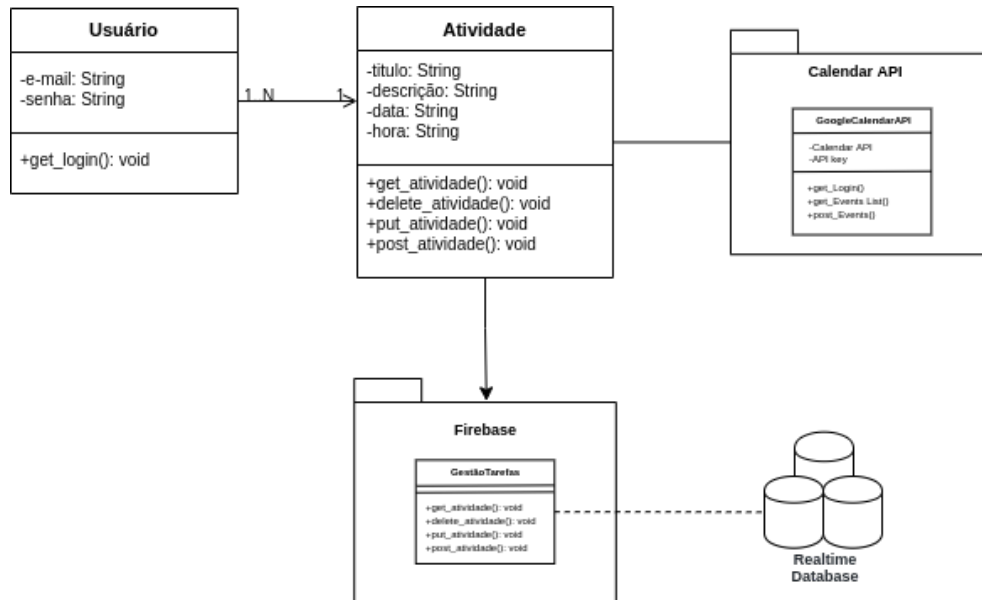


Fonte – Autoria própria

Após especificar cada caso de uso e demonstrar suas funcionalidades, foi elaborado um diagrama de classe. Este diagrama da figura 21 mostra, a estrutura das classes utilizadas no

domínio de negócio do sistema, apresentando seus componentes principais e a inter-relação entre eles. Também, mostra a associação existente entre Google Calendar<sup>3</sup>.

Figura 21 – Diagrama de classes dos protótipos



Fonte – Autoria própria

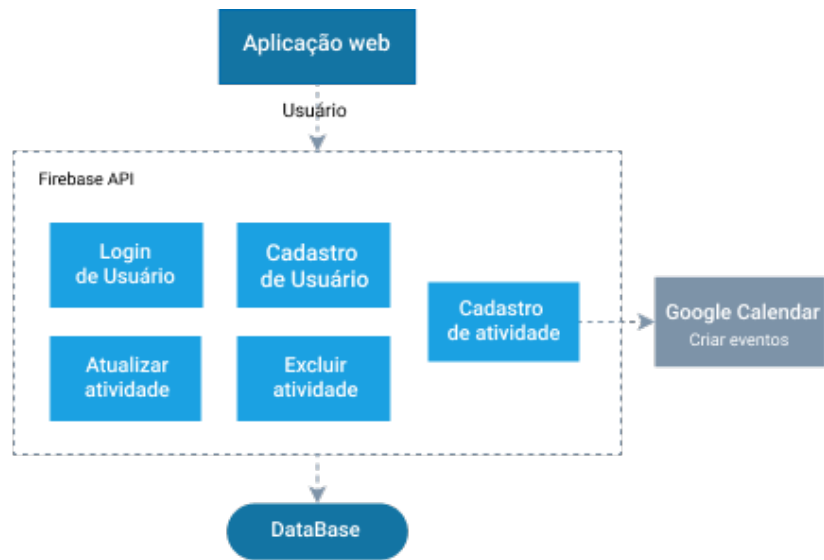
A figura 22 demonstra uma visualização do diagrama de componentes, representando sua modelagem de relacionamento entre os diferentes componentes do sistema, ampliando os serviços dentro da aplicação Firebase API. Na qual, o controlador (aplicação web) em Rest fornece pontos de acesso para a API, usando seus serviços para acessar o banco de dados.

A figura 23 apresenta um diagrama de atividade que contempla a descrição de um caso de cadastramento de uma atividade na aplicação. O processo inicia quando o sistema encontra uma chamada de cadastro e termina no momento em que o solicitante informa todas as informações da chamada. O usuário pode ser encaminhado através de dois caminhos principais, com ou sem a associação entre Google Calendar.

Com a compreensão dos requisitos e funcionalidades determinadas para a construção do sistema de controle de atividades, foi executada a representação visual de cada etapa da aplicação de forma mais detalhada, estabelecendo os componentes e informações do contexto real da aplicação. Também foram atribuídas propriedades visuais como cores, fontes, tamanhos e ícones.

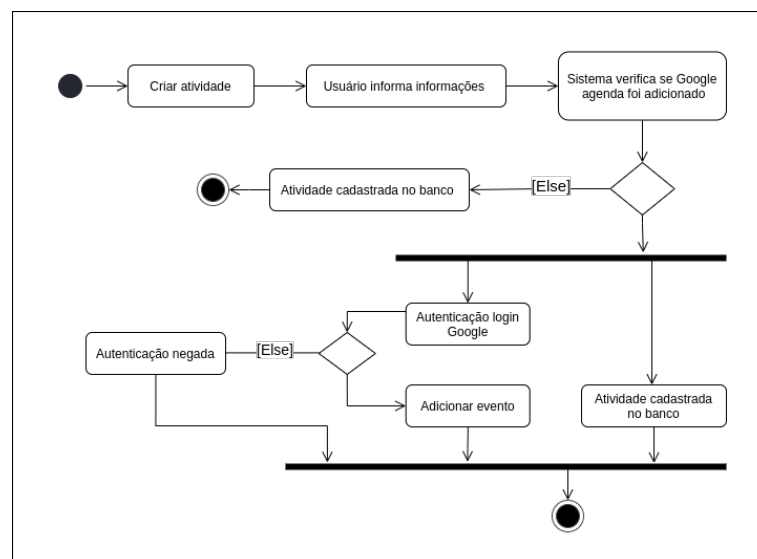
<sup>3</sup> <<https://developers.google.com/calendar>>

Figura 22 – Diagrama de componentes



Fonte – Autoria própria

Figura 23 – Diagrama de atividade



Fonte – Autoria própria

Figura 24 – Tela de login

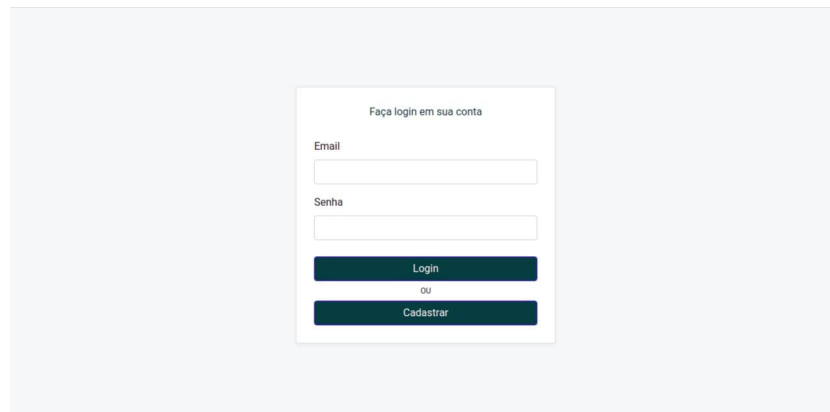
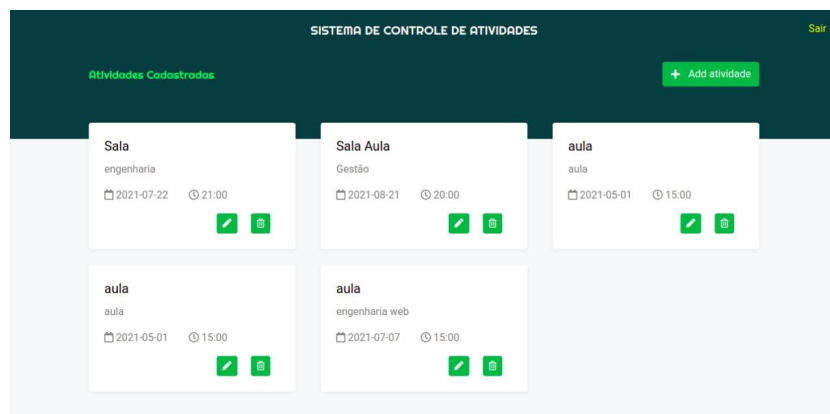


Figura 24 mostra a tela de login do sistema. O formulário contém o título "Faça login em sua conta", campos para "Email" e "Senha", e botões para "Login" e "Cadastrar".

Fonte – Autoria própria

Conforme figura 24, o usuário precisa estar logado no sistema para obter acesso aos demais recursos (cadastro, atualização ou exclusão de atividades). Os dados de autenticação solicitados são o Email e a Senha do usuário, na qual o processo de cadastro pode ser realizado na falta dessas informações de autenticação. Para fins de usabilidade o usuário pode visualizar mensagens padrões de erros de informações inseridas nos campos. Ao logar no sistema o usuário é redirecionado para a página inicial do sistema.

Figura 25 – Tela inicial



Fonte – Autoria própria

A visão base da aplicação apresentada na figura 25, é constituída de por dois componentes: Menu de navegação e Componente principal. O Menu de navegação apresenta a logo do sistema, um botão de logout da sessão de autenticação e um botão de cadastramento de atividades dentro da aplicação. O componente principal é o que será atualizado com todas as atividades já existentes no banco juntamente com as ações de atualização das informações sobre a atividade e a opção de excluir atividade.



Figura 26 – Tela de cadastro de atividade

Fonte – Autoria própria

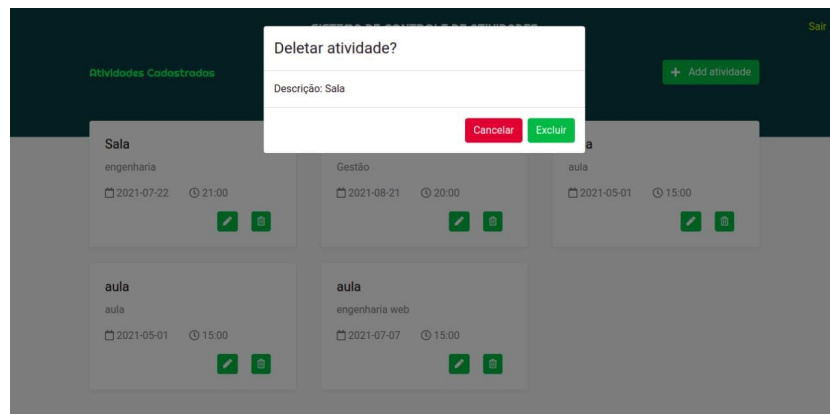
O formulário de cadastro de novas atividades, representado na figura 26, tem como requisito a inserção dos dados de cada atividade, tais como o Título, Descrição, Data e Hora de cada atividade. Possibilitando também a ativação da opção de vinculação com o Google Calendar, cadastrando tanto no banco local da aplicação, quanto nos eventos do Google Calendar.

Figura 27 – Tela de atualizar atividade

Fonte – Autoria própria

Para atualizar informações sobre uma atividade, exemplificada na figura 27, é recuperado os dados no banco de dados e passados para o formulário de edição, cada atividade instanciada individualmente através de um id. Dessa forma, quando o usuário clica em editar são mostrados na tela de edição de atividade, os dados relativos ao id de um atividade. O id também foi utilizado para excluir atividades na aplicação, mostrando uma tela de confirmação de exclusão ao usuário, conforme mostra a figura 28.

Figura 28 – Tela de deletar atividade



Fonte – Autoria própria

### 5.3 Design Patterns

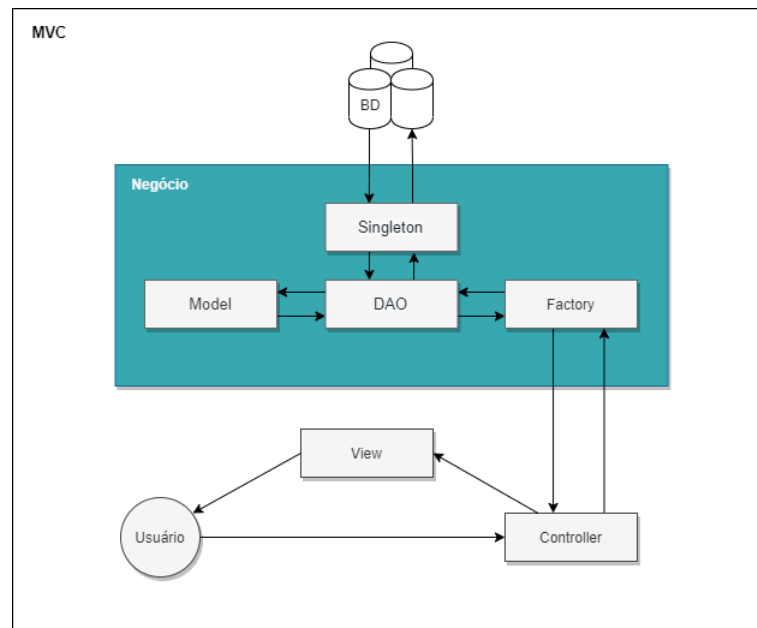
Assim como existem muitos *frameworks front-end* Web, o mesmo acontece com os *design patterns*. A seção 2.3 apresentou alguns deles, tanto dos padrões GoF, como arquitetural e da Web. Os *patterns* escolhidos para a implementação foram:

- **Singleton:** assegura que uma instância única de objeto seja criada em uma classe;
- **Factory:** encapsula a criação de objetos, permitindo que as subclasses decidam quais objetos criar;
- **Builder:** permite construir objetos complexos, separando sua produção em uma série de etapas;
- **MVC:** responsável por desacoplar a interface da navegação e regras de negócio;
- **DAO:** encapsula os mecanismos de acesso e persistência dos dados da lógica da aplicação.

A prova de conceito será apresentada na seção 5.2 e segue a estrutura base da construção constituída pelo MVC, o que permite dividir a aplicação em três componentes: Model; View; Controller. A figura 29 contém a modelagem utilizada pelo MVC, apresentando sua divisão e os *patterns* escolhidos para a implementação da aplicação.

O DAO será utilizado para acesso e persistência dos dados, na qual visa gerenciar a conexão com o banco de dados para obter e armazenar informações. Conforme apresentado na figura 30, a classe DAO fica responsável pelo acesso e persistência dos dados do cliente, separando a classe de negócio de classe de acesso aos dados da aplicação. A classe ControleAtividade da

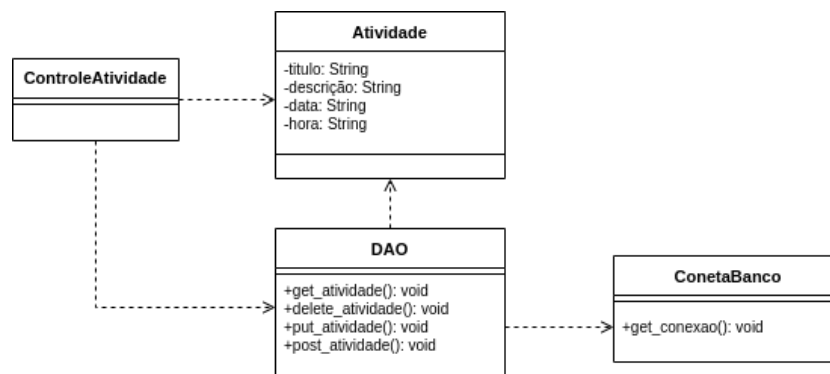
Figura 29 – Camadas da Arquitetura de Software.



Fonte – Autoria própria

prova de conceito é a classe do domínio que representa uma atividade.

Figura 30 – Relacionamento DAO.



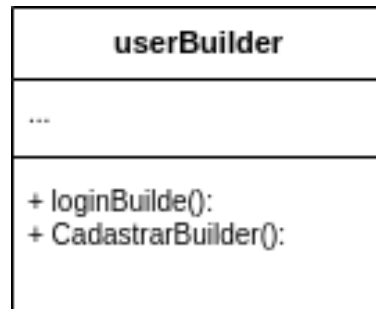
Fonte – Autoria própria

O Factory Method será empregado para fornecer um método de construção de objetos em uma superclasse, permitindo que as subclasses decidam qual classe instanciar.

Para diminuir a complexidade de construção dos objetos, foi empregado Builder, visando extrair o código de construção do objeto da própria classe, permitindo sua construção em uma série de etapas. O Builder é utilizado na construção do Usuário (ver figura 31).

O Singleton é implementado para acesso a autenticação e acesso ao banco de dados. Conforme a figura 32, é definido uma operação Instance que permite os clientes acessarem sua

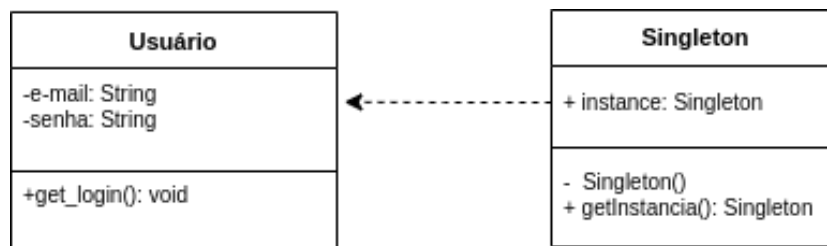
Figura 31 – Relacionamento Builder.



Fonte – Autoria própria

única instância, esta classe possui também um construtor de classe privado evitando que seja usado de novo e uma variável estática que salvará a pedido dessa classe.

Figura 32 – Relacionamento Singleton.



Fonte – Autoria própria

## 5.4 Métricas de Desempenho e Software

A fim de realizar a comparação entre as quatro versões da aplicação de controle de atividades (React e Vue.js, com e sem design patterns), foram selecionadas métricas de desempenho e software, visando analisar aspectos que afetam diretamente o desenvolvimento de aplicações, como a quantidade de código, curva de aprendizado e custo de execução quando aplicado *design pattern*.

As métricas escolhidas a nível de software para alimentar a análise de comparação estão presentes na tabela 5, tais métricas deverão ser adotadas para acompanhar o desenvolvimento dos sistemas, promovendo informações que evidenciem os pontos de melhoria ou impacto<sup>4</sup>.

A tabela 6, a seguir, contém as métricas de desempenho escolhidas para a análise das implementações, a fim de identificar quais tendências estão sendo alcançadas e verificar se

<sup>4</sup> <[https://www.ic.unicamp.br/~eliane/Cursos/RSM/rsm\\_descricao.htm](https://www.ic.unicamp.br/~eliane/Cursos/RSM/rsm_descricao.htm)>

Tabela 5 – Métricas de software

<b>Métrica</b>	<b>Descrição</b>
Complexidade	Curva de aprendizagem e suporte da comunidade
Quantidade de linhas de código	Quantidade de código necessário para a mesma aplicação
Quantidade de linhas em branco	Quantidade de linhas em branco para a mesma aplicação
Número de funções desenvolvidas	Quantidade de funções desenvolvidas para a mesma aplicação
Número de arquivos do projeto	Quantidade de arquivos para a mesma aplicação
Tamanho do build	Comparar o tamanho do build gerado

Fonte – Autoria própria

os critérios estabelecidos identificaram melhorias nos processos.

Tabela 6 – Métricas de desempenho

<b>Métrica</b>	<b>Descrição</b>
Consumo de memória	Verificar o consumo obtido nas duas aplicações
Consumo de CPU	Verificar o consumo obtido nas duas aplicações
Tempo de renderização	Medir o tempo necessário para o carregamento da página
Tempo de resposta	Medir o tempo necessário realizar uma requisição cliente-servidor na aplicação

Fonte – Autoria própria

Assim, os experimentos realizados devem permitir avaliar os diferentes resultados encontrados na execução das versões da implementação da prova de conceito, visto que para uma mesma aplicação, os *frameworks* e as bibliotecas testadas podem ter seus resultados distintos quando aplicados os padrões.

## 5.5 Considerações Finais

Esse capítulo apresentou o trabalho proposto, exibindo os *frameworks* e *design patterns* adotadas para o desenvolvimento da prova de conceito e as métricas de desempenho e software que serão utilizadas para a análise comparativa. O próximo capítulo será apresentado os experimentos conduzidos para análise e quais foram os resultados encontrados.

## 6 RESULTADOS

Este capítulo apresenta os testes de desempenho realizados nas versões implementadas da prova de conceito e a análise comparativa dos resultados. A seção 6.1 apresenta o plano de teste realizado no experimento. A seção 6.2 aponta os diferentes resultados encontrados na execução dos experimentos em cada um dos *frameworks* escolhidos, com e sem *design patterns*. Por fim, a seção 6.3 faz uma comparação entre os trabalhos relacionados e o proposto.

### 6.1 Plano de Testes

O plano de testes para o experimento consiste em realizar testes de desempenho e coleta de métricas de software. As métricas escolhidas foram apresentadas na proposta. A coleta das métricas tem por objetivo obter uma base de análise comparativa entre a implementação da prova de conceito com e sem *design pattern*, possibilitando avaliar os diferentes resultados encontrados no desenvolvimento da aplicação, verificando a viabilidade da utilização de *patterns* e apresentar o *framework* com melhor desempenho computacional e menor impacto gerado.

Para realizar a comparação dos *frameworks* escolhidos, foi utilizado o ambiente de desenvolvimento com a seguinte configuração:

- Linux ubuntu 24.10
- Intel Core i5 7200U CPU @ 2.50GHz
- 8GB RAM
- Google Chrome 91

Para a coleta de métricas de software, foram utilizadas de ferramentas de terceiros. Foram realizadas coletas de quantidade de linhas de código, linhas em branco, o número de funções desenvolvidas, tamanho do build, complexidade funcional e o número de arquivos gerados.

Para realizar a contagem de linhas de código e linhas em branco será utilizado uma ferramenta de análise JavaScript chamada DeepScan<sup>1</sup>, inspecionando todo o código JavaScript de forma abrangente e computando o número total. Foi realizado um build nativo nos *frameworks*

---

<sup>1</sup> <<https://deepscan.io/>>

para gerar o conjunto de arquivos JavaScript compilados, a fim de mensurar o tamanho de cada aplicação. Para medir a Complexidade funcional será utilizado o plugin Code Metrics<sup>2</sup>.

Para a coleta de métricas de desempenho, foram realizadas coletas de consumo de memória e CPU, tempo de renderização e tempo de resposta. Para as três primeiras foi utilizado a ferramenta de desenvolvedor do Google Chrome. Para o tempo de resposta foi realizado a marcação do *timestamp* no início de uma requisição até o término da mesma. Foram realizadas 50 coleta, excluindo os 10 piores e 10 melhores resultados, resultando em um total de 30 coletas, pois de acordo com o Teorema Central do Limite (TCL) (FISCHER, 2010), quando se tem uma amostra suficientemente grande, a distribuição de probabilidade da média amostral pode ser aproximada por uma distribuição normal. Essa aproximação é válida a partir de 30 médias amostrais.

## 6.2 Resultados Encontrados

A coleta destas informações foram organizadas de forma comparativa, estabelecendo indicadores que ajudem a verificar os impactos dos *patterns* no desenvolvimento.

### 6.2.1 Métricas de Software

Conforme tabela 7 e 8, pode-se chegar a algumas conclusões que ajudem a estabelecer marcos referenciais nos projetos analisados, apresentando os resultados encontrados na utilização de *design patterns* com relação as métricas de software.

Tabela 7 – Métricas de software sem design pattern

Métricas de software	React	Vue.js
Linhas de Código (LOC)	506	502
Linhas em Branco	141	273
Tamanho do build	202,75 Kb	300,06 kb
Número de arquivos	10	10
Número de funções	11	11
Complexidade funcional	29	29

Fonte – Autoria própria

De acordo com a tabela 7, é possível verifica que sem *design patterns* o Vue.js apresenta mais linhas em branco e um tamanho de build maior que o React.

<sup>2</sup> <<https://github.com/kisstkondoros/codemetrics>>

Tabela 8 – Métricas de software com design pattern

Métricas de software	React	Vue.js
Linhas de Código (LOC)	688	690
Linhas em Branco	200	370
Tamanho do build	203,287 kb	301,74 kb
Número de arquivos	15	15
Número de funções	40	40
Complexidade funcional	78	79

Fonte – Autoria própria

De acordo com a tabela 8, é possível verificar que com *design patterns*, a aplicação sofreu um aumento em todas métricas de software avaliadas. O Vue.js continua apresentando uma maior quantidade de linhas em branco e um tamanho de build do que o React. Em decorrência da aplicação de *patterns*, o tamanho de build sofreu um aumento de 0,26% para React e 0,55% para Vue.js, assim, aumento de uma versão para a outra foi de 2,15 vezes maior. Também é possível analisar que quando aplicado *design patterns*, a quantidade de arquivos aumentou em 50% para ambos os *frameworks*. Outro resultado encontrado é que a complexidade computacional aumentou consideravelmente (cerca de 168,96%).

Logo, é possível afirmar que aplicar *design patterns*, aumenta a quantidade de linhas de código, linhas em branco, tamanho do build, número de arquivos, números de funções e a complexidade funcional, e conseqüentemente tem um impacto negativo no desenvolvimento de aplicações Web em termos das métricas de software avaliadas.

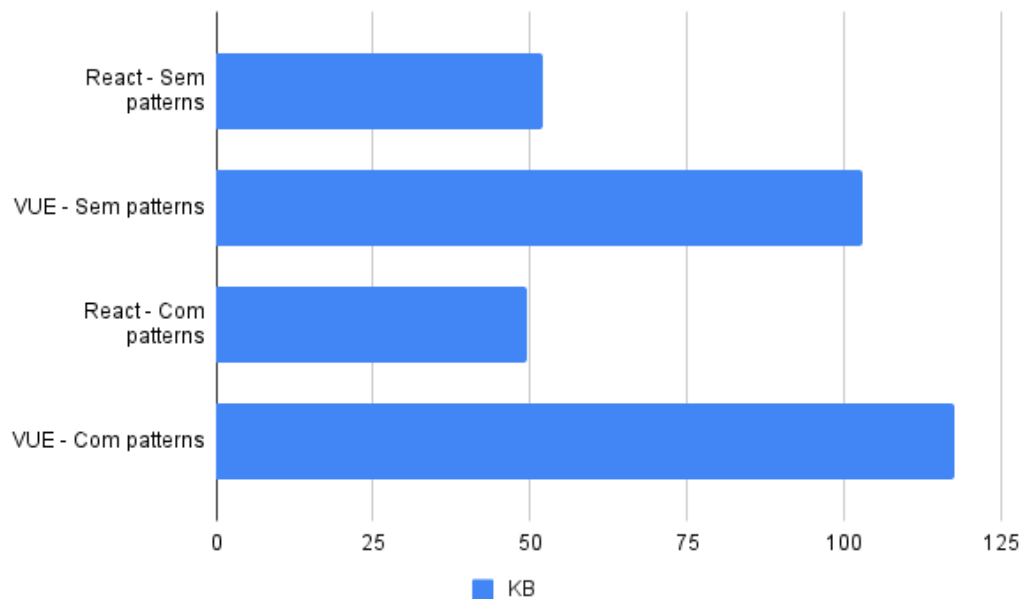
### 6.2.2 Métricas de Desempenho

Para a obtenção dos dados de memória, CPU e renderização, foi utilizado cada versão da aplicação durante 15 segundos e analisado com a ferramenta de desenvolvedor do Google Chrome.

A figura 33, apresenta os resultados encontrados na média das execuções dos testes para o consumo de memória RAM. Conclui-se que a implementação React quando aplicou os *design patterns* teve um melhor desempenho quando comparado a versão sem padrão. O Vue.js retratou um consumo mais elevado quando aplicado. Assim, conclui-se que aplicar *design pattern* melhora o desempenho da aplicação em termos de consumo de memória quando utiliza o *framework* React, mas piora quando utiliza o Vue.js.



Figura 33 – Resultado dos testes para consumo de memória



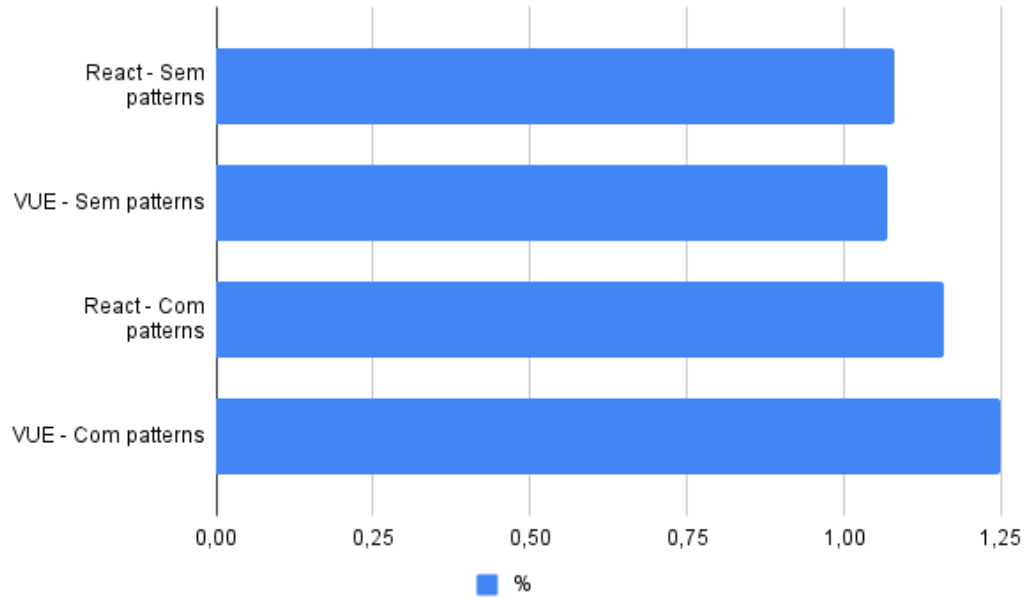
Fonte – Autoria própria

A figura 34 ilustra os resultados encontrados nas médias dos testes para consumo de CPU, detalhando o consumo específico de cada versão. Conclui-se que os *framework* desenvolvidos de forma pura apresentaram um consumo menor de CPU quando comparados a versão com *design patterns*. O Vue.js apresentou um aumento mais elevado do que o React quando aplicado *patterns*. Assim, conclui-se que aplicar *desing pattern* piora o desempenho da aplicação em termos de consumo de CPU, principalmente quando utiliza o Vue.js.

A figura 35 ilustra os resultados encontrados na execução dos testes para tempo de renderização. Conclui-se que os *framework* desenvolvidos com *design patterns* aumentaram o tempo de renderização. O Vue.js apresentou um menor aumento quando comparado ao React. Assim, conclui-se que aplicar *desing pattern* piora o desempenho da aplicação em termos de tempo de renderização, principalmente quando utiliza o React.

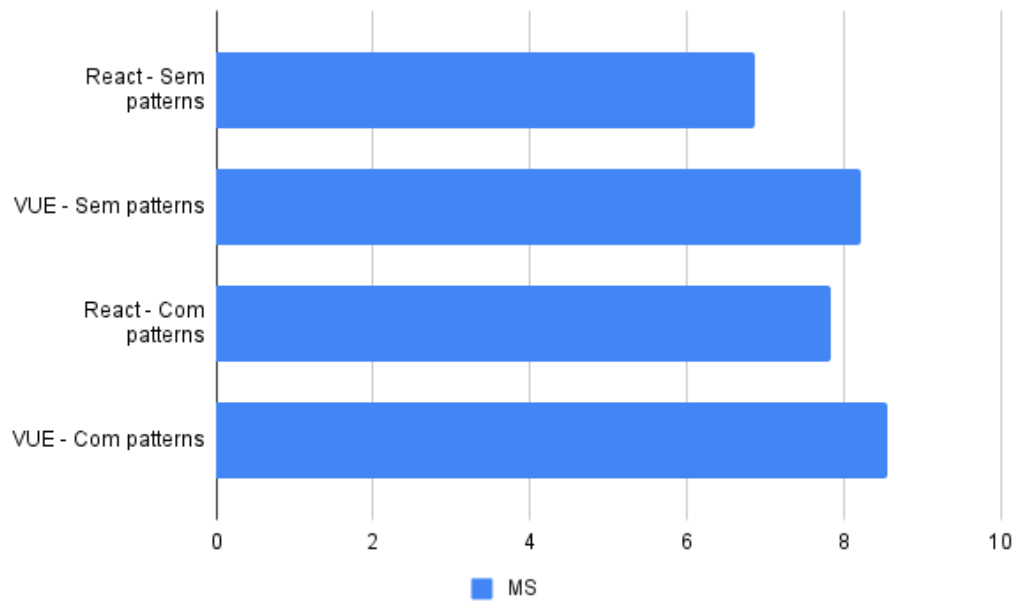
As figuras 36, 37, 38, 39 e 40 ilustram os resultados encontrados na execução dos testes para tempo de resposta, considerando a quantidade de tempo que leva para efetuar uma operação por completo. Foi adotado a função `Date.now()` para obter o *timestamp* da operação. É possível verificar que quando aplicado *design patterns*, tanto a versão com React como com Vue.js apresentaram um tempo de resposta menor. Assim, conclui-se que aplicar *desing pattern* melhora o desempenho da aplicação em termos de tempo de resposta.

Figura 34 – Resultado dos testes para consumo de CPU.



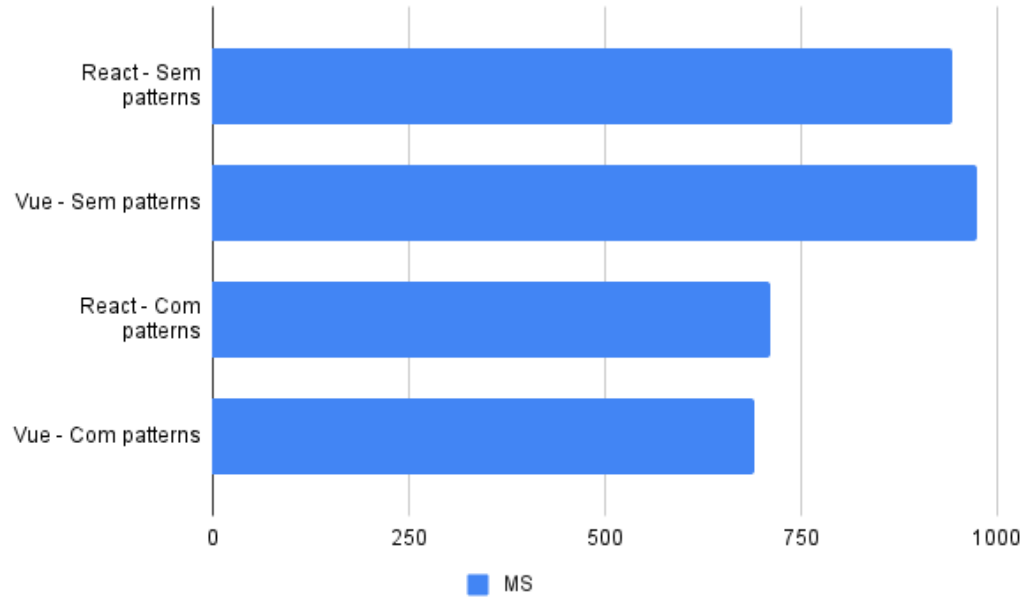
Fonte – Autoria própria

Figura 35 – Resultado dos testes para Tempo de renderização.



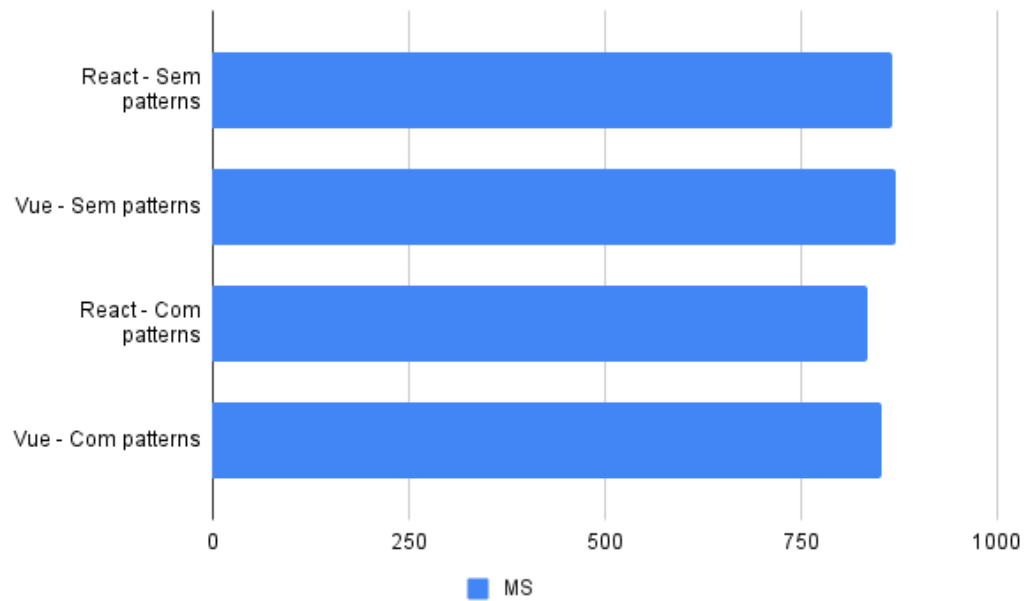
Fonte – Autoria própria

Figura 36 – Resultado dos testes para Tempo de resposta - Login de Usuário.



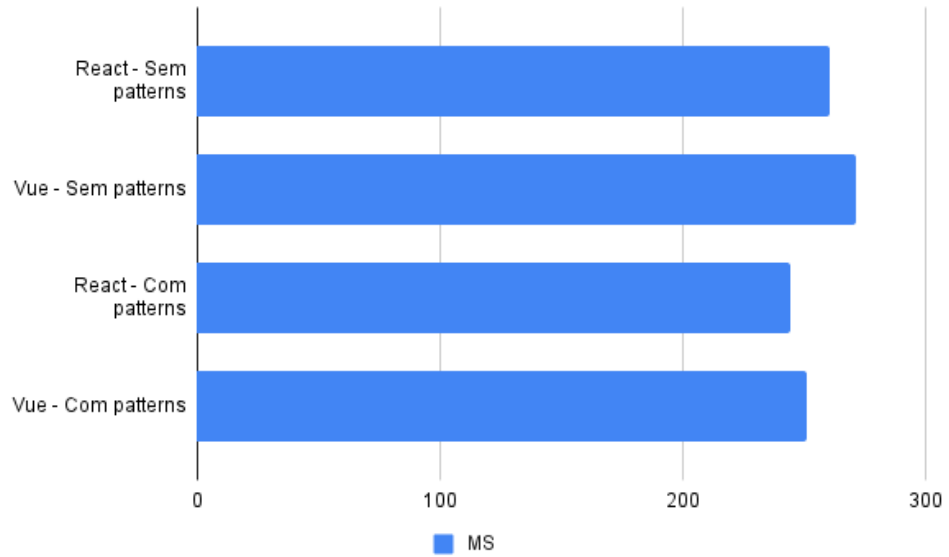
Fonte – Autoria própria

Figura 37 – Resultado dos testes para Tempo de resposta - Cadastro de Usuário.



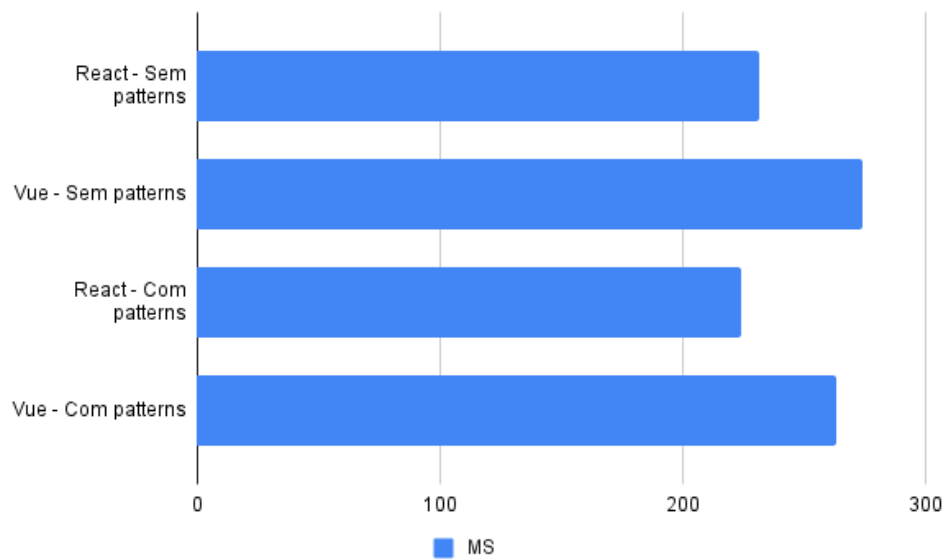
Fonte – Autoria própria

Figura 38 – Resultado dos testes para Tempo de resposta - Cadastro de atividade.



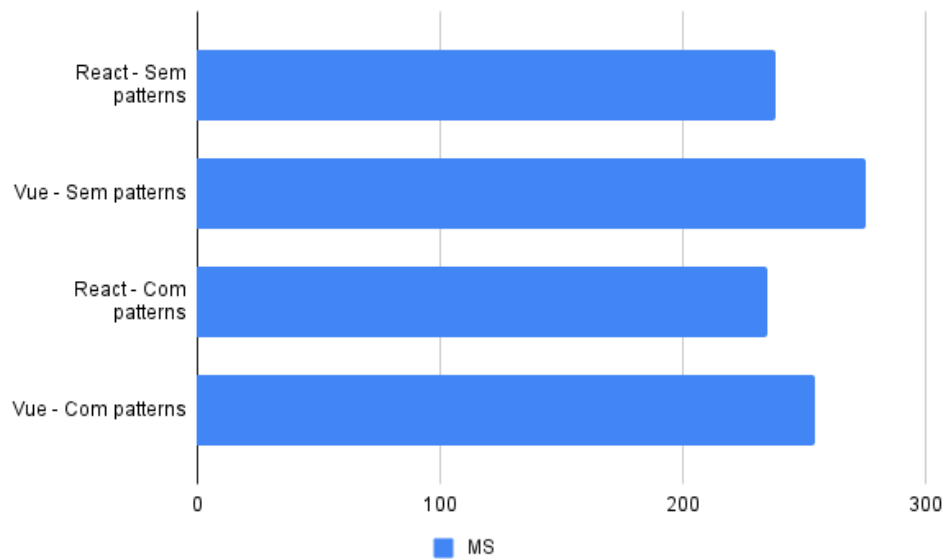
Fonte – Autoria própria

Figura 39 – Resultado dos testes para Tempo de resposta - Atualizar atividade.



Fonte – Autoria própria

Figura 40 – Resultado dos testes para Tempo de resposta - Excluir atividade.



Fonte – Autoria própria

A tabela 9 apresenta os dados referentes aos gráficos apresentados anteriormente em relação ao Tempo de resposta.

Tabela 9 – Tempo de resposta comparando a versão sem e com padrão

Operações	React	Vue.js	React   patterns	Vue.js   patterns
Login de Usuário	941,724	973,586	711,172	689,551
Cadastro de Usuário	866,275	871,758	833,79	853,103
Cadastro de atividade	260,517	271,068	244,31	250,793
Atualizar atividade	231,172	274,068	223,965	263,31
Excluir atividade	238,103	275,724	235,068	254,206

Fonte – Autoria própria

### 6.3 Comparativo Entre os Trabalhos

A fim de caracterizar o trabalho proposto e comparar com os trabalhos relacionados, a tabela 10 detalha o comparativo entre eles:

É possível verificar a partir da tabela 10 que o trabalho proposto possui uma análise com métricas de desempenho, métricas de software e realiza a comparação do antes e depois de aplicado os *design patterns*.

Tabela 10 – Comparativo Entre os Trabalhos Relacionados e a Proposta

Trabalho	Métricas de Desempenho	Métricas de Software	Comparação Antes / Depois	Quantidade de Padrões
Cechinel et al.	Sim	Sim	Sim	3
Mantoan	Sim	Não	Sim	5
Thung et al	Sim	Não	Sim	7
Trabalho proposto	Sim	Sim	Sim	5

Fonte – Autoria própria

#### 6.4 Considerações Finais

Esse capítulo apresentou os testes realizados e os resultados encontrados com a coleta das métricas de desempenho e software. Foi verificado que utilizar *design patterns* impactou negativamente nas métricas de software. Quando comparado em termos de desempenho, aplicar *design pattern* pode melhorar em alguns aspectos, como por exemplo, tempo de resposta e consumo de memória. No próximo capítulo será apresentado algumas conclusões que ajudem a verificar a viabilidade, os impactos obtidos, as dificuldades e limitações encontradas, além da identificação de possíveis para trabalhos futuros.

## 7 CONCLUSÃO

Atualmente, torna-se cada vez mais comum o desenvolvimento de grandes aplicações que produzem diversas combinações de ferramentas e técnicas para o desenvolvimento em abordagem Web. A partir dessa necessidade, a presente pesquisa visou apresentar a viabilidade da utilização de *design patterns* em *frameworks front-end* a partir da implementação de aplicações Web.

O trabalho proposto apresentou uma fundamentação teórica essencial para uma pré-  
via compreensão de todos os conceitos e elementos fundamentais que relacionam a engenharia de software, a evolução web, *design patterns* e *frameworks* JavaScript, a fim de realizar uma investigação sobre o campo de interesse da pesquisa e apresentar definições para o desenvolvimento do trabalho.

Foram apresentados trabalhos relacionados que fazem o uso da abordagem proposta nesta pesquisa, verificando seus processos de desenvolvimento, critérios e escolhas de análise para compreender as melhores alternativas para o trabalho.

Foi apresentado uma metodologia para a realização da pesquisa, exibindo a distribuição das etapas adotadas: levantamento bibliográfico; definição dos *frameworks*, o desenvolvimento do estudo de caso; a seleção dos *design patterns*; definição das métricas de software e métricas de desempenho; realização dos experimentos nos *frameworks* e análise comparativa entre os resultados encontrados.

O trabalho proposto apresentou também uma proposta de realizar uma comparação entre *frameworks* front-end quando aplicado *design patterns*, visando a realização de um estudo com o objetivo de obter os resultados acerca da implementação de aplicação Web com e sem a utilização dos *design patterns* em *frameworks front-end JavaScript*.

Além disso, foram apresentados os resultados encontrados da análise comparativa em termos de desempenho e métricas de software, quando aplicado os *design patterns* nos *frameworks*. Pode-se concluir que em termos de software, eles possuem poucas diferenças significativas entre *frameworks* no geral. Possuem similaridade na quantidade de linhas de código, linhas em branco, tamanho do build gerado, número de arquivos, número de funções

e complexidade computacional ocasionado principalmente por serem baseados na mesma linguagem. Quando aplicado *patterns*, os resultados mostram um aumento de todas as métricas, principalmente em termos de número de arquivos e complexidade computacional.

Quando comparado em termos de desempenho, tanto de consumo de cpu e memória, como tempo de renderização e resposta, foi verificado que melhora o desempenho da aplicação em termos de consumo de memória quando utiliza o *framework* React, mas piora quando utiliza o Vue.js. Em relação a CPU, piora o desempenho da aplicação, principalmente quando utiliza o Vue.js. Para a renderização, piora o desempenho da aplicação, principalmente quando utiliza o React. Por fim, se for considerado o tempo de resposta, melhora o desempenho. Como conclusão, pode-se destacar que além dos *design patterns* trazer diversos benefícios de reusabilidade, legibilidade e manutenção, a fim de prolongar seu ciclo de vida ao construir uma arquitetura de software padronizada, o trabalho proposto mostrou que depende do que é avaliado para definir se melhora ou piora quando aplica *design patterns* em aplicações que utilizam *frameworks* front-end Web.

## 7.1 Limitações

Algumas dificuldades e limitações foram encontradas no decorrer da pesquisa, quanto a tecnologias e ferramentas utilizadas que tiveram que ser estudadas durante o desenvolvimento do trabalho. Além da implementação da prova de conceito com e sem *design pattern* ter levado muito tempo do escopo do trabalho, acarretando na não construção da aplicação no *framework* Angular. Outra dificuldade encontrada foi construção e aplicação dos *designs patterns* nos *frameworks* JavaScript escolhidos por conterem na sua arquitetura diversos padrões de software.

## 7.2 Trabalhos Futuros

Propõe-se para possíveis de trabalhos futuros:

- Um relato de experiência sobre como foi implementação e utilização dos *design patterns* no trabalho proposto;
- Estudar e aplicar novos *patterns* não abordados, para uma maior avaliação;
- Estudar e aplicar outras métricas de testes para avaliar diferentes pontos de desempenho dos *frameworks*;



- Implementar a prova de conceito em outros *frameworks*, a fim de verificar o desempenho e métricas de software em outras tecnologias; e
- Construção de uma aplicação mais robusta, possibilitando analisar seus diferentes resultados ao aumentar a complexidade do sistema.

## REFERÊNCIAS

- AFONSO, A. **O que é Angular?** 2018. Disponível em: <<https://blog.algaworks.com/o-que-e-angular/>>. Acesso em: 13 Mar.2021.
- AGHAEI, S.; NEMATBAKHSH, M. A.; FARSANI, H. K. Evolution of the world wide web: From web 1.0 to web 4.0. **International Journal of Web & Semantic Technology**, Academy & Industry Research Collaboration Center (AIRCC), v. 3, n. 1, p. 1–10, 2012.
- BALLE, A. R. Análise de metodologias ágeis: conceitos, aplicações e relatos sobre xp e scrum. 2011.
- BECK, K.; BEEDLE, M.; BENNEKUM, A. v.; COCKBURN, A.; CUNNINGHAM, W.; FOWLER, M.; THOMAS, D. Manifesto para desenvolvimento ágil de software. **Retirado em**, v. 20, 2001.
- BOMFIM, M. N. d. C.; SAMPAIO, F. F. A web 2.0, suas tecnologias e aplicações educacionais. **Relatório Técnico NCE**, Brasil, 2008.
- CASTELEYN, S.; DANIEL, F.; DOLOG, P.; MATERA, M. **aplicativos da web de engenharia**. [S.l.: s.n.], 2009. v. 30.
- CECHINEL, A. *et al.* Avaliação do framework angular e das bibliotecas react e knockout para o desenvolvimento do frontend de aplicações web. Florianópolis, SC, 2017.
- CHOUDHURY, N. World wide web and its journey from web 1.0 to web 4.0. **International Journal of Computer Science and Information Technologies**, Citeseer, v. 5, n. 6, p. 8096–8100, 2014.
- CHRISTOPHER ALEXANDER ISHIKAWA, S.; SILVERSTEIN, M. *et al.* A pattern language. **Oxford University Press**, Oxford University Press, 1977.
- CONTE, T.; MENDES, E.; TRAVASSOS, G. H. Processos de desenvolvimento para aplicações web: Uma revisão sistemática. In: **Proceedings of the 11th Brazilian Symposium on Multimedia and Web (WebMedia 2005)**. [S.l.: s.n.], 2005. v. 1, p. 107–116.
- DUARTE, A. R. Metodologia rails: análise da arquitetura model view controller aplicada. Universidade Federal de Minas Gerais, 2011. Disponível em: <<http://hdl.handle.net/1843/BUOS-94MMY9>>. Acesso em: 23 fev.2021.
- DUARTE, N. F. B. **Frameworks e Bibliotecas Javascript**. Tese (Doutorado) — Instituto Superior De Engenharia Do porto, 2015.
- FAYAD, M. E.; SCHMIDT, D. C.; JOHNSON, R. E. **Building application frameworks: object-oriented foundations of framework design**. [S.l.]: John Wiley & Sons, Inc., 1999.
- FILHO, D. L. B. Experiências com desenvolvimento ágil. **São Paulo**, 2008.
- FILIPOVA, O. **Learning Vue. js 2**. [S.l.]: Packt Publishing Ltd, 2016.
- FISCHER, H. **A history of the central limit theorem: From classical to modern probability theory**. [S.l.]: Springer Science & Business Media, 2010.

- FOWLER, M. **Patterns of Enterprise Application Architecture**. [S.l.]: New Jersey: Prentice Hall, 2002. v. 1.
- FRANCO, E. F. **Um modelo de gerenciamento de projetos baseado nas metodologias ágeis de desenvolvimento de software e nos princípios da produção enxuta**. 2007. Dissertação (Mestrado em Sistemas Digitais) — Universidade de São Paulo, São Paulo, 2007.
- FREEMAN, E.; FREEMAN, E.; SIERRA, K.; BATES, B. **Use A Cabeça Padrões E Projetos**. [S.l.]: Alta Books, 2009.
- GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Padrões de Projetos: Soluções Reutilizáveis de Software Orientados a Objetos**. [S.l.]: Bookman, 2000. v. 1.
- GARLAN, D.; SHAW, M. An introduction to software architecture. Carnegie Mellon University, 1994.
- GERALDES, J. T.; MIRANDA, R. P. d. Infra-estrutura para desenvolvimento e implementação de aplicações web em java utilizando máquina de estados. Brasília, DF, 2002.
- GIL, A. C. Como classificar as pesquisas. **Como elaborar projetos de pesquisa**, Atlas São Paulo, v. 4, p. 44–45, 2002.
- GORLA, J. P. F.; FOSCHINI, I. J. Arquitetura para desenvolvimento web baseado em jsf 2.0 utilizando padrões de projeto. **Revista TIS**, v. 2, n. 3, 2014.
- KEYES, J. **Software engineering handbook**. [S.l.]: CRC Press, 2002.
- KULESZA, R.; SOUSA, M. F. de; LIMA, M.; ARAUJO, C.; FILHO, M. A. Evolução das arquiteturas de software rumo à web 3.0. **Sociedade Brasileira de Computação**, 2018.
- LEFF, A.; RAYFIELD, J. T. Web-application development using the model/view/controller design pattern. In: IEEE. **Proceedings fifth ieee international enterprise distributed object computing conference**. [S.l.], 2001. p. 118–127.
- LEMOES, M. F. de; OLIVEIRA, P. C.; RUELA, L. C.; SANTOS, M. da S.; SLVEIRA, T. C.; REIS, J. C. de S. Aplicabilidade da arquitetura mvc em uma aplicação web (webapps). **RE3C-Revista Eletrônica Científica de Ciência da Computação**, v. 8, n. 1, 2013.
- LINO, C. E. **Reestruturação de software com adoção de padrões de projeto para a melhoria da manutenibilidade**. 2011. Monografia (Bacharel em Sistemas de Informação), URCAMP (Universidade Federal de Lavras), Lavras.
- MALDONADO, J. C.; BRAGA, R. T. V.; GERMANO, F. S. R.; MASIERO, P. C. Padrões e frameworks de software. **Notas Didáticas, Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, ICMC/USP, São Paulo, SP, Brasil**, 2002.
- MANTOAN, F. G. Proposta de arquitetura de desenvolvimento web baseada em php utilizando design patterns. um estudo de caso. **CENTRO DE ENSINO SUPERIOR DE FOZ DO IGUAÇU**, 2009.
- MARCONI, M. d. A.; LAKATOS, E. M. **Fundamentos de metodologia científica**. [S.l.]: 5. ed.-São Paulo: Atlas, 2003.

- MORALES-CHAPARRO, R.; LINAJE, M.; PRECIADO, J.; SÁNCHEZ-FIGUEROA, F. Mvc web design patterns and rich internet applications. **Proceedings of the Jornadas de Ingeniería del Software y Bases de Datos**, p. 39–46, 2007.
- NING, W.; LIMING, L.; YANZHANG, W.; YI-BING, W.; JING, W. Research on the web information system development platform based on mvc design pattern. In: IEEE. **2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology**. [S.l.], 2008. v. 3, p. 203–206.
- PEREIRA, A. d. S. **PADRÕES DE PROJETO: UMA COMPILAÇÃO DOS MAIS UTILIZADOS EM PROJETOS DE SOFTWARE**. 2008. Monografia (Bacharel em Sistemas de Informação), URCAMP (FACULDADE DE MINAS), FAMINAS-BH.
- PONTES, T. B.; ARTHAUD, D. D. B. Metodologias ágeis para o desenvolvimento de softwares. **Ciência E Sustentabilidade**, v. 4, n. 2, p. 173–213, 2018.
- PRESSMAN, R. S. **Engenharia de Software**. [S.l.]: McGrawHill Editora, 2006.
- PRESSMAN, R. S. **Engenharia de Software: uma abordagem profissional**. [S.l.]: McGrawHill Editora, 2011.
- ROPELATO, P. P. O. T. **RECONSTRUÇÃO DE UMA APLICAÇÃO WEB UTILIZANDO PADRÕES ÁREA: Engenharia de Software. Palavras-chave: Design patterns. WebPatterns**. Tese (Doutorado) — UNIVERSIDADE REGIONAL DE BLUMENAU, 2007.
- SAKS, E. Javascript frameworks: Angular vs react vs vue. 2019.
- SARDAGNA, M.; VAHLDICK, A. Aplicação do padrão data access object (dao) em projetos desenvolvidos com delphi. **Escola Regional de Banco de Dados**, 2008.
- SCHWABER, K. **Agile project management with Scrum**. [S.l.]: Microsoft press, 2004.
- SCUR, C. Uma revisão das implementações do front controller e uma proposta de ajuste. UNIVERSIDADE FEEVALE, 2012.
- SHALLOWAY, A.; TROTT, J. R. **Explicando Padroes de Projetos**. [S.l.]: Bookman, 2004.
- SOMMERVILLE, I. **Engenharia de Software. 9ª Edição, 2011. ed.** [S.l.]: Pearson Education–BR, 2011.
- SRIDARAN, R.; PADMAVATHI, G.; IYAKUTTI, K. A survey of design pattern based web applications. **J. Object Technol.**, v. 8, n. 2, p. 61–70, 2009.
- TAKEUCHI, H.; NONAKA, I. The new new product development game. **Harvard business review**, v. 64, n. 1, p. 137–146, 1986.
- THUNG, P. L.; NG, C. J.; THUNG, S. J.; SULAIMAN, S. Improving a web application using design patterns: A case study. In: IEEE. **2010 International Symposium on Information Technology**. [S.l.], 2010. v. 1, p. 1–6.
- VIDAL, D. F. Dao–data access object: Uma aplicação prática do padrão de mapeamento objeto relacional. INTEP, 2016.
- VORA, P. **Web application design patterns**. [S.l.]: Morgan Kaufmann, 2009.

## APÊNDICE A – IMPLEMENTAÇÃO DA PROVA DE CONCEITO

Esse apêndice refere-se a implementação da prova de conceito, apresentando os códigos dos padrões aplicados. Os códigos apresentados são para o *frameworks* React. A versão Vue.js pode ser verificado no repositório: <<https://github.com/henriquecatunda/Projeto-TCC>>

O View possui apenas as representações das informações de forma visual, no desenvolvimento foram aplicados os recursos de interface de comunicação com usuário, dispondo e guardando todas as informações. A Listagem 1 refere-se a View de listagem de atividades, contendo apenas trechos de código JavaScript simples para percorrer dados e viabilizar estes dados.

### Listagem 1 – Camada View - Listagem

```

1
2 <Container fluid="as" className="main">
3   <div className="sairLogin" onClick={() => SairUser()}>Sair</div>
4   <div className="boxTop">
5     <div className="tituloEmpresa">Sistema de controle de atividades</div>
6
7     <Navbar className="navigation justify-content-between" >
8       <Navbar.Brand className="descricaoEmpresa"> Atividades cadastradas </
9         Navbar.Brand>
10      <Form inline>
11        { /* <FormControl type="text" placeholder="Pesquisar" className="
12          mr-sm-2" /> */}
13        <ModalCadastro/>
14      </Form>
15    </Navbar>
16  </div>
17
18  <div className="boxContainer">
19
20    {listTarefas &&
21      listTarefas.map((atividade, index) => (
22        <Card style={{ width: '20rem', border: 'none', marginBottom: '10px' }}
23          key={index}>
24          <Card.Body style={{ padding: '1.5rem' }}>
25            <Card.Title className="bodyTitle">{atividade.title}</Card.Title>
26            <Card.Text className="formTitle">
27              {atividade.description}
28            </Card.Text>
29            <Card.Text>
30              <div className="formDateTime">

```

```

28         <div>
29             <i class="far fa-calendar"></i> {atividade.date}
30         </div>
31         <div>
32             <i class="far fa-clock"></i> {atividade.time}
33         </div>
34     </div>
35 </Card.Text>
36 <Col xs={5} className="groupBotaoStructure" >
37     <ModalAtualizar id={atividade.key} tarefa={atividade}/>
38     <ModalExcluir id={atividade.key} descricao={atividade.title
39         }/>
40 </Col>
41 </Card.Body>
42 </Card>
43     )})
44 </div>
</Container>

```

Conforme Listagem 2, é exibido o código de um Controller, definindo todas as ações que o usuário pode requisitar, visando intermediar tais requisições solicitadas pelo View e repassando para as camadas posteriores.

### Listagem 2 – Camada Controller

```

1  import DataService from "../Service/service";
2
3  class DataController {
4
5      loginUser = (email,password) => {
6          return DataService.loginValidar(email,password);
7      }
8
9      cadastrarUser = (email,password) => {
10         return DataService.CadastrarValidar(email,password);
11     }
12
13     observadorUser = (user) => {
14         return DataService.observadorValidar(user);
15     }
16
17     sairLogin = () => {
18         return DataService.sairValidar();
19     }
20
21     getAll = (setListTarefas) => {

```

```

22     return DataService.getAllValidar(setListTarefas);
23 }
24
25 cadastrarAtividade = (tarefa) => {
26     return DataService.cadastrarAtividadeValidar(tarefa);
27 }
28
29 cadastrarGoogle = (event) => {
30     return DataService.cadastrarGoogleValidar(event);
31 }
32
33 atualizarAtividade = (key,value) => {
34     return DataService.atualizarAtividadeValidar(key,value);
35 }
36
37 deletarAtividade = (key) => {
38     return DataService.deletarAtividadeValidar(key);
39 }
40
41 }
42
43 export default new DataController();

```

Para manter uma separação e isolar as preocupações de lógica de validação da camada Controller, foi adicionado uma camada de serviço. Conforme mostrado na Listagem 3, todos os campos requeridos pela operação serão verificados seguindo a lógica de negócios da aplicação.

### Listagem 3 – Camada Service

```

1 import taskDAO from "../DAO/dao";
2
3 class DataService {
4
5     loginValidar(email,password){
6         if (email.length === 0 || password.length === 0) {
7             alert('Informar todos os campos!');
8         }
9
10        if(typeof email !== "undefined"){
11            let lastAtPos = email.lastIndexOf('@');
12            let lastDotPos = email.lastIndexOf('.');
13
14            if (!(lastAtPos < lastDotPos && lastAtPos > 0 && email.indexOf('@@') == -1 &&
15                lastDotPos > 2 && (email.length - lastDotPos) > 2)) {
16                alert('Email não valido. ');
17            }
18        }
19    }
20 }

```

```
18     return taskDAO.login(email,password);
19 }
20
21 CadastrarValidar(email,password){
22     if (email.length === 0 || password.length === 0) {
23         alert('Informar todos os campos!');
24     }
25
26     if(typeof email !== "undefined"){
27         let lastAtPos = email.lastIndexOf('@');
28         let lastDotPos = email.lastIndexOf('.');
29
30         if (!(lastAtPos < lastDotPos && lastAtPos > 0 && email.indexOf('@@') == -1 &&
31             lastDotPos > 2 && (email.length - lastDotPos) > 2)) {
32             alert('Email não valido. ');
33         }
34     }
35     return taskDAO.inserir(email,password);
36 }
37
38 observadorValidar(user){
39     if (typeof user !== "undefined" ) {
40         alert('usuário invalido');
41     }
42     return taskDAO.observador(user);
43 }
44
45 sairValidar(){
46     return taskDAO.sair();
47 }
48
49 getAllValidar(setListTarefas){
50     return taskDAO.get(setListTarefas);
51 }
52
53 cadastrarAtividadeValidar(tarefa){
54     if (typeof tarefa !== "object" ) {
55         alert('cadastro não efetuado, atividade invalida');
56     }
57
58     return taskDAO.create(tarefa);
59 }
60
61 cadastrarGoogleValidar(event){
62     if (typeof event !== "undefined" ) {
63         alert('cadastro não efetuado, evento invalido');
64     }
65 }
```



```

66     return taskDAO.createGoogle(event);
67
68   }
69
70   atualizarAtividadeValidar(key,value){
71     if (typeof value !== "undefined" ) {
72       alert('Atualização não efetuada, campos inválidos');
73     }
74
75     return taskDAO.update(key,value);
76   }
77
78   deletarAtividadeValidar(key){
79     return taskDAO.delete(key);
80   }
81
82 }
83 export default new DataService();

```

O DAO foi adotado para separar as regras de acesso a banco de dados, funcionando como um intermediador entre a aplicação e o banco de dados, agrupando seu acesso por similaridade, fornecendo todos os métodos DAO específicos, sendo responsável pela leitura ou gravação no banco de dados. A Listagem 4 ilustra melhor o funcionamento da estrutura DAO.

#### Listagem 4 – Pattern DAO

```

1  import { firebase } from '@firebase/app'
2  import "firebase/auth";
3  import database from "../firebase";
4  import { google } from "../Config/config";
5
6  const db = database;
7
8
9  class DataDAO {
10
11    login = (email,password) => {
12      return firebase.auth().signInWithEmailAndPassword(email, password);
13    }
14
15
16    inserir = (email,password) => {
17      return firebase.auth().createUserWithEmailAndPassword(email, password);
18    }
19
20

```

```
21 observador = (user) => {
22   return firebase.auth().onAuthStateChanged((user) => {
23     if (user) {
24       var uid = user.uid;
25     } else {
26     }
27   });
28 }
29
30
31 sair = () => {
32   return firebase.auth().signOut().then(() => {
33     console.log("logout com sucesso!");
34   }).catch((e) => {
35     console.log(e);
36   });
37 }
38
39
40 get = async (setListTarefas) => {
41   var user = firebase.auth().currentUser;
42   if (user) {
43     let listTarefa = [];
44
45     const results = await db.get();
46     results.forEach(result => {
47       let key = result.key;
48       let data = result.val();
49       listTarefa.push({
50         key: key,
51         title: data.title,
52         description: data.description,
53         date: data.date,
54         time: data.time,
55       });
56     });
57     return setListTarefas(listTarefa);
58   }
59   console.log("sem usuario");
60 }
61
62
63 create(tarefa) {
64   return db.push(tarefa);
65 }
66
67
68 createGoogle(event) {
69   google.gapi.load('client:auth2', () => {
```

```

70 console.log('loaded client')
71
72 google.gapi.client.init({
73   apiKey: google.API_KEY,
74   clientId: google.CLIENT_ID,
75   discoveryDocs: google.DISCOVERY_DOCS,
76   scope: google.SCOPEES,
77 })
78
79 google.gapi.client.load('calendar', 'v3', () => console.log('bam!'))
80 google.gapi.auth2.getAuthInstance().signIn()
81 .then(() => {
82
83   var request = google.gapi.client.calendar.events.insert({
84     'calendarId': 'primary',
85     'resource': event,
86   })
87
88   request.execute(event => {
89     console.log(event)
90     window.open(event.htmlLink)
91   })
92 })
93 })
94 }
95
96
97 update(key, value) {
98   return db.child(key).update(value);
99 }
100
101
102 delete(key) {
103   return db.child(key).remove();
104 }
105 }
106
107 export default new DataDAO();

```

O pattern Factory Method foi adotado para delegar a criação de objetos, permitindo manter o controle de criação pelo seu tipo instanciado. O código da Factory é mostrado na Listagem 5, possuindo o método userFactory() que permite o cliente instruir o tipo de usuário a ser criado, passando seu argumento de tipo para o método de Fábrica.

```

1 function userFactory(email,password,type){
2
3     let factory = {};
4
5     if(type === 'user'){
6         factory.email = email;
7         factory.password = password;
8     }
9
10    if(type === 'dev'){
11        factory.email = email;
12        factory.password = password;
13    }
14
15    return factory;
16 }

```

Para garantir a conexão ao banco de dados por meio de uma instância, foi implementado o pattern Singleton. Conforme apresentado na Listagem 6, A instância FirebaseDatabase retorna uma mesma referência em chamadas subsequentes, lidando com o acesso de seus dados.

#### Listagem 6 – Pattern Singleton

```

1 import { firebase } from '@firebase/app'
2 import "firebase/database";
3 import "firebase/auth";
4
5 var firebaseConfig = {
6     apiKey: "XXX",
7     authDomain: "gestao-tarefas-40c08.firebaseio.com",
8     projectId: "gestao-tarefas-40c08",
9     storageBucket: "gestao-tarefas-40c08.appspot.com",
10    messagingSenderId: "546393215838",
11    appId: "1:546393215838:Web:e909604f81fcb89549f5cf",
12    databaseURL: "https://gestao-tarefas-40c08-default-rtdb.firebaseio.com/",
13 };
14
15 firebase.initializeApp(firebaseConfig);
16
17 var database = firebase.database().ref("/tarefas");
18
19 export default database;

```