UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Lucas Abrantes Furtado Silva

INVESTIGAÇÃO DO USO DE PADRÕES ARQUITETURAIS EM PROJETOS QUE UTILIZAM REACT E ANGULAR

CAMPINA GRANDE - PB

2023

Lucas Abrantes Furtado Silva

INVESTIGAÇÃO DO USO DE PADRÕES ARQUITETURAIS EM PROJETOS QUE UTILIZAM REACT E ANGULAR

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: João Arthur Brunet Monteiro

CAMPINA GRANDE - PB

2023

Lucas Abrantes Furtado Silva

INVESTIGAÇÃO DO USO DE PADRÕES ARQUITETURAIS EM PROJETOS QUE UTILIZAM REACT E ANGULAR

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

João Arthur Brunet Monteiro

Orientador – UASC/CEEI/UFCG

Melina Mongiovi Brito Lira

Examinadora – UASC/CEEI/UFCG

Francisco Vilar Brasileiro

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 28 de junho de 2023.

CAMPINA GRANDE - PB

RESUMO

No desenvolvimento de software, a forma como o frontend é desenvolvido tem passado por uma evolução significativa, impulsionada pela introdução de novos frameworks e pela padronização de componentes, visando aprimorar a manutenibilidade, escalabilidade e a experiência do usuário. Considerando esse cenário, existem diversas opções de estratégias, formas de implementação e outras características a serem consideradas. Portanto, adotar um padrão permite fácil adaptação e modificações no sistema, melhorando a comunicação e colaboração entre as equipes e reduzindo a probabilidade de falhas. No entanto, não há estudos conhecidos que identifiquem os padrões arquiteturais mais utilizados, a forma como são implementados pelos desenvolvedores, seus desafios, em quais tipos de projeto um determinado padrão se encaixa melhor e quais os custos para implementá-los. O propósito deste trabalho é investigar a aplicação de padrões arquiteturais no frontend de múltiplos projetos, a fim de caracterizar os padrões arquiteturais prevalentes, bem como identificar e caracterizar suas possíveis variações. O objetivo final é contribuir para a identificação e propagação dos padrões arquiteturais mais aplicados no desenvolvimento do frontend utilizando React e Angular como frameworks.

INVESTIGATION OF THE USE OF ARCHITECTURAL PATTERNS IN PROJECTS USING REACT AND ANGULAR

ABSTRACT

In software development, the way the frontend is developed has undergone a significant evolution, driven by the introduction of new frameworks and the standardization of components, aiming to improve maintainability, scalability and user experience. Considering this scenario, there are several strategy options, forms of implementation and other characteristics to be considered. Therefore, adopting a standard allows for easy adaptation and modifications to the system, improving communication and collaboration between teams and reducing the likelihood of failures. However, there are no known studies that identify the most used architectural patterns, the way they are implemented by developers, their challenges, which types of projects a given pattern fits best and what the costs are to implement them. The purpose of this work is to investigate the application of architectural patterns in the frontend of multiple projects, in order to characterize the prevalent architectural patterns, as well as identify and characterize their possible variations. The final objective is to contribute to the identification and propagation of the most applied architectural patterns in frontend development using React and Angular as frameworks.

Investigação do Uso de Padrões Arquiteturais em Projetos que Utilizam React e Angular.

Lucas Abrantes Furtado Silva
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
lucas.abrantes.silva@ccc.ufcg.edu.br

João Arthur Brunet Monteiro
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
joao.arthur@computacao.ufcg.edu.br

RESUMO

No desenvolvimento de *software*, a forma como o *frontend* é desenvolvido tem passado por uma evolução significativa, impulsionada pela introdução de novos *frameworks* e pela padronização de componentes, visando aprimorar a manutenibilidade, escalabilidade e a experiência do usuário. Considerando esse cenário, existem diversas opções de estratégias, formas de implementação e outras características a serem consideradas. Portanto, adotar um padrão permite fácil adaptação e modificações no sistema, melhorando a comunicação e colaboração entre as equipes e reduzindo a probabilidade de falhas. No entanto, não há estudos conhecidos que identifiquem os padrões arquiteturais mais utilizados, a forma como são implementados pelos desenvolvedores, seus desafios, em quais tipos de projeto um determinado padrão se encaixa melhor e quais os custos para implementá-los. O propósito deste trabalho é investigar a aplicação de padrões arquiteturais no *frontend* de múltiplos projetos, a fim de caracterizar os padrões arquiteturais prevalentes, bem como identificar e caracterizar suas possíveis variações. O objetivo final é contribuir para a identificação e propagação dos padrões arquiteturais mais aplicados no desenvolvimento do *frontend* utilizando React e Angular como *frameworks*.

PALAVRAS-CHAVE:

Frontend, software, padrões arquiteturais,

desenvolvimento.

1. INTRODUÇÃO

Com o constante avanço do desenvolvimento de *software,* a rápida evolução das tecnologias e o grande aumento da competitividade entre as empresas, a demanda de projetos na área aumenta a cada ano, fazendo com que empresas invistam cada vez mais nesse setor, contratando novos desenvolvedores para suprir a demanda de desenvolvimento. O lançamento bem-sucedido de novos produtos ou serviços no mercado é crucial para a sobrevivência a longo prazo de uma empresa.

Um dos quesitos mais importantes é garantir que as funcionalidades estejam de acordo com o que foi solicitado pelo cliente. Levando esse fato em consideração, é possível constatar que o código no *frontend* está se tornando cada vez mais complexo, demandando cada vez mais o uso de padrões apropriados. O *frontend*, provavelmente por ter que lidar com a experiência e a interface gráfica do usuário (podendo variar consideravelmente de acordo com o dispositivo, o tamanho da tela e as preferências do usuário), possui uma carência de técnicas bem estabelecidas. Porém, com a utilização de *frameworks* mais conhecidos, como Angular e React, esse problema é amenizado e até resolvido, desde que usados da forma correta e com o padrão adequado.

Ter um padrão arquitetural bem definido é de extrema importância em um projeto. Isso permite que os desenvolvedores reduzam a sobrecarga cognitiva, economizem tempo na tomada de decisões e melhorem a comunicação com a equipe [2]. Além disso, um padrão arquitetural facilita a integração de novos desenvolvedores, pois lidam com um código estruturado e bem definido.

O objetivo é alcançar uma arquitetura limpa, conceito esse que tem como foco principal a divisão de interesses, realizando a separação do *software* em camadas, de acordo com suas regras de negócio e interfaces. Assim, a aplicação se torna independente de estruturas, testável, com uma interface do usuário que pode ser facilmente alterada sem impactar no restante do sistema e independente de banco de dados, podendo substituí-los sem afetar a regra de negócios.

Dentre os padrões arquiteturais mais conhecidos, podemos citar o MVC, MVP, MVVM, Flux e Micro Frontend. Com uma ampla variedade de opções de padrões, é necessário saber qual escolher. A dificuldade frequente que os desenvolvedores encontram ao identificar padrões no *frontend* é evidenciada pela escassez de projetos com uma estrutura bem definida.

Portanto, o objetivo desta investigação é identificar os padrões arquiteturais empregados em projetos que utilizam os *frameworks* React e Angular. Serão identificados os padrões mais comumente adotados, analisando se estão sendo aplicados de forma adequada e identificando os motivos que levaram à escolha desses padrões arquiteturais.

A seguir, na seção 2, faremos uma breve introdução de conceitos importantes para o melhor entendimento deste trabalho. Na seção 3, detalharemos os métodos utilizados para a obtenção dos resultados e como os dados foram analisados. Na seção 4, teremos os resultados, apresentando quais os padrões arquiteturais mais utilizados em projetos que utilizam React e Angular e o que motivou o uso desses padrões. Por fim, na seção 5, trataremos as conclusões e trabalhos futuros, falando sobre os resultados e os possíveis métodos de investigação quem podem ser usados em trabalhos posteriores.

2. FUNDAMENTAÇÃO TEÓRICA

A seção a seguir apresenta a fundamentação teórica para a compreensão deste artigo. Encontram-se descritos abaixo os principais conceitos identificados durante a revisão bibliográfica.

2.1 Principais Conceitos

Padrão Arquitetural: É uma solução prévia, estudada, testada e documentada para um problema comum no desenvolvimento de *software* [3]. Ela fornece orientação na tomada de decisões, levantando questões sobre a utilidade, funções e relacionamentos de cada subsistema, estabelecendo assim a estrutura fundamental do *software*. A arquitetura ajuda a definir a base sólida sobre a qual o *software* será construído, proporcionando um roteiro claro para o desenvolvimento e facilitando a compreensão e manutenção do sistema ao longo do tempo.

FrontEnd: Em termos simples, o *frontend* de uma aplicação é a camada visual com a qual o usuário interage [4]. O desenvolvedor *frontend* é responsável por criar uma experiência agradável e intuitiva para o usuário, utilizando elementos que definem a identidade visual do *software*. As linguagens comumente utilizadas nessa tarefa são HTML, CSS, JavaScript e Typescript. O *frontend* é responsável por apresentar os dados e funcionalidades de forma organizada e acessível, garantindo uma interface amigável e responsiva para o usuário.

MVC: O padrão de arquitetura de *software* conhecido como MVC tem como objetivo principal otimizar a velocidade das requisições realizadas pelos usuários [5]. Ele é composto por três componentes essenciais: *Model* (Modelo), *Controller* (Controle) e *View* (Visão). A camada *Model* é responsável por gerenciar o comportamento dos dados, aplicando funções, lógica e regras de negócio estabelecidas. Ela armazena os dados recebidos do *Controller*, realiza validações e retorna as respostas adequadas. O *Controller* atua como intermediário entre as requisições enviadas pela *View* e as respostas fornecidas pelo *Model*. Ele processa os dados informados pelo usuário e os repassa para as demais camadas. Já a camada *View* é responsável por apresentar visualmente as informações ao usuário, sendo o ponto de atuação do desenvolvedor *frontend*. Ela é a interface de comunicação direta com o usuário, responsável por enviar as solicitações ao *Controller* e exibir as respostas obtidas do sistema. Em resumo, a *View* disponibiliza e captura as informações do usuário. O MVC é amplamente utilizado devido a diversos benefícios, como maior segurança, organização, eficiência, economia de tempo e flexibilidade na manutenção do código, se necessário. Podemos observar o fluxo de eventos desse padrão arquitetural na Figura 1.

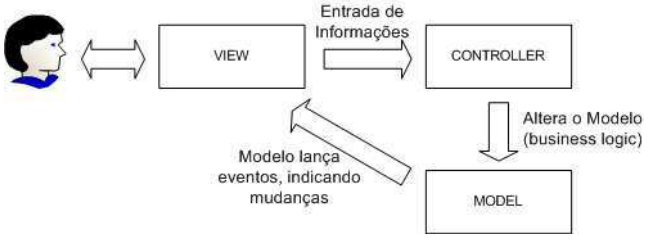

FIGURA 1. Fluxo de eventos e informações em uma arquitetura MVC.

MVP: O padrão Model View Presenter (MVP) é especialmente útil quando se busca uma clara separação das camadas lógicas de uma aplicação, isolando a visualização das demais camadas. Esse padrão também facilita a reutilização de código, principalmente quando é necessário implementar um mesmo recurso em diferentes tecnologias. Essas características permitem que os desenvolvedores trabalhem de forma mais independente. No MVP, a camada *Model* abrange os dados, com suas classes de domínio e regras de negócio. Na camada *View*, assim como no padrão MVC, encontra-se a parte de visualização, com elementos de interface gráfica e interação com o usuário. Já o *Presenter* é responsável pela apresentação dos dados e atua como o intermediário entre a *View* e o *Model*. No MVP, a comunicação entre as camadas *View-Presenter* e *Presenter-Model* ocorre por meio de interfaces. A relação entre *Presenter* e *View* é de um-para-um, e as classes *Model* e *View* não possuem conhecimento uma da existência da outra [6]. A figura 2 ilustra o fluxo de eventos desse padrão arquitetural.

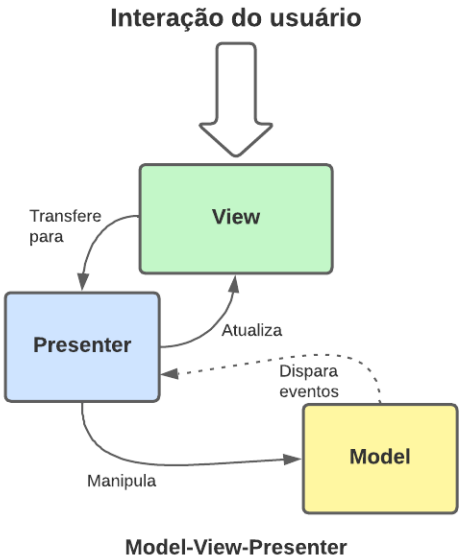

FIGURA 2. Fluxo de eventos e informações em uma arquitetura MVP.

MVVM: No padrão Model View ViewModel (MVVM), as camadas *View* e *Model* permanecem como antes. A diferença principal é a adição da camada *ViewModel* (VM), que coordena as operações entre a *View* e o *Model*, além de realizar operações específicas sobre o *Model* quando necessário. Esse padrão é amplamente utilizado no desenvolvimento *mobile* e possui pontos fortes, como facilidade de aprendizado e adaptação ao padrão, possibilidade de um estilo mais iterativo e exploratório, simplificação dos testes de unidade e facilidade na manutenção do código. Na prática, o MVVM é uma especialização do MVP, pois são bastante semelhantes, mas com a distinção de que o MVVM sugere separar a lógica de apresentação de dados da lógica principal de negócios do aplicativo, enquanto o MVP é mais independente de plataforma [7]. O fluxo de eventos desse padrão arquitetural é ilustrado na Figura 3.

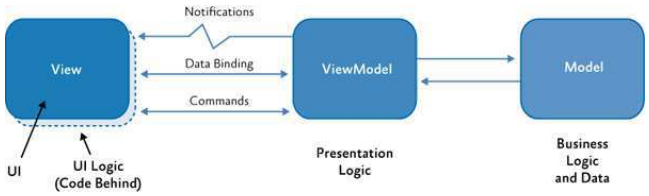

FIGURA 3. Fluxo de eventos e informações em uma arquitetura MVVM.

Micro FrontEnd: A ideia por trás desse padrão é aplicar os conceitos de microsserviços ao *frontend*, enxergando a aplicação web como uma composição de funcionalidades que são propriedades de equipes independentes. Cada equipe se especializa em uma área de negócio específica e tem autonomia para escolher e atualizar sua própria *stack*, sem a necessidade de alinhamento com outras equipes. O compartilhamento de tempo de execução não é recomendado, mesmo que as equipes do projeto utilizem o mesmo *framework*, pois a aplicação deve ser independente, evitando o uso de estado compartilhado ou variáveis globais [8]. Uma convenção de nomes pode ser estabelecida para situações em que o isolamento não é possível. Além disso, as funcionalidades devem ser úteis mesmo que a linguagem utilizada falhe ou não esteja em execução. A ilustração do fluxo de eventos desse padrão arquitetural pode ser observada na Figura 4.

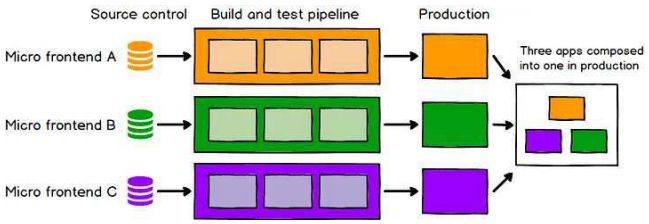

FIGURA 4. Fluxo de eventos e informações em uma arquitetura de Micro FrontEnd.

Flux: Conhecido por ser um padrão introduzido pelo Facebook, Flux é um modelo de gerenciamento de estado unidirecional projetado inicialmente para solucionar problemas de complexidade e escalabilidade no desenvolvimento do *frontend*. O Flux foi desenvolvido para funcionar juntamente com *frameworks* e bibliotecas, em especial React e Redux, embora também seja possível utilizá-lo com outras tecnologias. Devido ao seu fluxo unidirecional de dados, as atualizações de estado da aplicação acontecem em uma única direção, fazendo com que o rastreamento de alterações e depurações se tornem mais fáceis [9]. De modo geral, apesar de existirem variações nas especificidades da implementação, o estado é mantido em *Stores* centralizados, facilitando a consistência e a gestão do estado global. Na Figura 5, é possível visualizar de forma clara e representativa o fluxo de eventos desse padrão arquitetural.

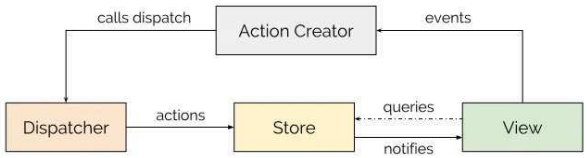

FIGURA 5. Fluxo de eventos e informações em uma Flux.

Component-Based Architecture : Também conhecida como Arquitetura baseada em Componentes, é um padrão arquitetural que procura trazer organização e estrutura para o desenvolvimento da aplicação, utilizando componentes reutilizáveis. Nessa abordagem, a aplicação é dividida em componentes independentes, que são desenvolvidos, testados e versionados. Quando reutilizados, esses componentes promovem a modularidade e a fácil manutenção no código. É um padrão fácil de ser implementado, já que a maioria dos *frameworks* atuais oferecem recursos específicos para facilitar a criação e o gerenciamento de componentes [10]. É um padrão que assim como o Flux e o Micro Frontend, por exemplo, utiliza componentes, só que nesse caso, de forma mais simplificada. A representação visual do fluxo de eventos desse padrão arquitetural é apresentada na Figura 6.

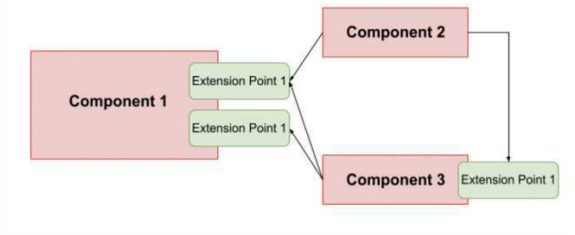

FIGURA 6. Fluxo de eventos e informações em uma arquitetura baseada em componentes.

3. METODOLOGIA

Este trabalho é caracterizado como um estudo de caso exploratório, abrangendo uma abordagem qualitativa e quantitativa. O objetivo principal deste artigo é reunir experiências de estudos variados em uma área específica, com o propósito de alcançar uma meta de pesquisa particular. O foco é

explorar de maneira metódica e estruturada o conhecimento já existente em campos correlatos. A finalidade principal é evidenciar informações, métricas e direções, por meio da avaliação de repositórios relacionados ao assunto.

Utilizamos os passos metodológicos a seguir: a elaboração das indagações de pesquisa; pesquisa nos repositórios do Github por meio das palavras-chave escolhidas; avaliação da qualidade dos códigos presentes nesses repositórios através de critérios de implementação, analisando a organização das pastas e arquivos no *frontend,* observando se os mesmos são implementados seguindo as diretrizes de um determinado padrão arquitetural, tornando-os de fácil entendimento para quem o observa, além de facilitar a reusabilidade do código; coleta dos dados e análise dos resultados.

No próximo tópico abordaremos as questões que este artigo se propõe a responder.

3.1 Questões de Pesquisa

- Quais os padrões arquiteturais mais utilizados em projetos que adotam React como ferramenta de desenvolvimento?
- Quais os padrões arquiteturais mais utilizados em projetos que adotam Angular como ferramenta de desenvolvimento?
- Esses padrões são aplicados da forma correta? Qual o principal motivo de utilizá-los?

3.2 Seleção de Estudos Primários

Na etapa de coleta dos dados, as pesquisas foram realizadas através de buscas no Github, utilizando as seguintes palavras-chave : “Project React Frontend" e “Project Angular Frontend”.

Durante a fase de seleção dos projetos, foram estabelecidos pré-requisitos para escolher os repositórios adequados. Consequentemente, alguns projetos encontrados no escopo da pesquisa foram excluídos da investigação. Os critérios estabelecidos nessa fase e utilizados para a seleção dos repositórios foram:

- Ser um projeto com pelo menos 4 *forks* no github. (Indicando uma certa relevância do projeto e que outras pessoas se interessaram no mesmo).
- Projeto iniciado a partir de 2018 (Visando analisar projetos recentes, de até 5 anos. Assim podemos descartar com mais facilidade projetos que usam linguagens ou *frameworks* mais antigos).
- Documentação transparente e de fácil entendimento.

Ao término da fase de seleção, obteve-se uma coleção de 820 projetos investigados, dos quais apenas 76 cumpriram todos os critérios. Os projetos que foram selecionados possuem uma média de 29135 linhas de código (Considerando apenas o código do *frontend*), sendo em sua grande parte em JavaScript, Typescript, CSS e HTML. Para fazer a contagem dessas linhas, foi utilizado a extensão VSCodeCounter. Ainda sobre os 76 projetos que cumpriram os critérios, observa-se uma média de 4 contribuidores por projeto. Podemos ver mais detalhadamente a relação de aceitação dos repositórios na Figura 7.

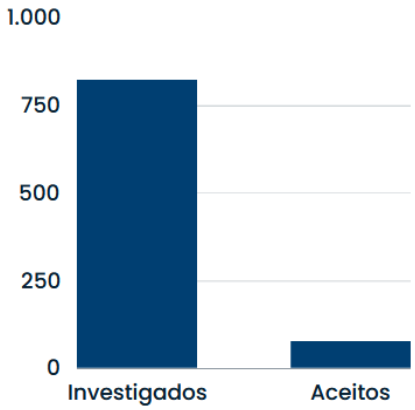

FIGURA 7. Relação dos repositórios selecionados.

Dos 76 repositórios analisados que cumpriram os critérios, 46 utilizam React no desenvolvimento do *frontend*, enquanto os demais optaram pelo Angular. Entre os projetos React aceitos, todos adotaram Javascript como linguagem principal de desenvolvimento. Já no caso do Angular, somente 3 dos 30 projetos escolheram Javascript, enquanto todos os demais optaram pelo uso de Typescript. A Figura 8 oferece uma visualização mais detalhada da relação entre os frameworks e as linguagens utilizadas, permitindo uma melhor compreensão desse contexto.

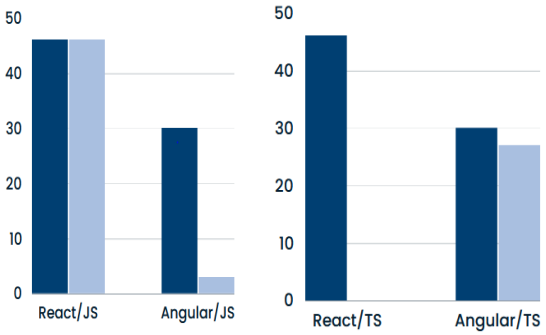

FIGURA 8. Relação entre frameworks e linguagens.

É importante ressaltar também que alguns projetos embora tenham sido classificados dentro de algum padrão (MVC, MVVM, Micro Frontend, Flux ou Component-Based), sofrem variações. Sendo elas :

- Variações na estrutura de pastas: A organização das pastas e arquivos no *frontend* pode variar de projeto para projeto.
- Variações nas convenções de nomenclatura e estilos de código
- Variações na integração com APIs e serviços externos: A forma como o *frontend* da aplicação se integra com APIs pode variar, envolvendo assim o uso de bibliotecas específicas, configurações de autenticação e diferentes

formatos de comunicação.

Sendo assim, caracterizamos como projetos de padrão arquitetural definido aqueles que cumprem rigorosamente as características de um padrão, ou seja:

- Atendem à risca a estrutura de pastas do padrão arquitetural, sem variações. Com a separação de responsabilidades bem definida.
- Adesão às diretrizes do padrão, aplicando corretamente (como manda o padrão) as convenções de nomenclatura e conceitos estabelecidos.

Já os projetos que apresentam um padrão arquitetural indefinido são precisamente aqueles que sofrem variações, não se assemelham em nada com um padrão arquitetural, ou até mesmo mescla características de mais de um padrão, ou seja :

- Possui uma estrutura de arquivos e pastas modificadas de acordo com a preferência dos desenvolvedores e não seguindo como sugere o padrão arquitetural escolhido pelo time.
- A forma como os componentes se comunicam no projeto não condizem com um padrão arquitetural específico.
- A forma como o *frontend* se integra com a API não segue nenhum padrão arquitetural.

Ao analisar exclusivamente os projetos classificados como tendo um padrão definido, constatamos que a maioria (71,05%) apresenta uma quantidade de linhas de código inferior à média (29135 linhas). Isso sugere que projetos maiores, com maior número de linhas de código, enfrentam dificuldades na adoção de um padrão. A Figura 9 oferece uma visão mais detalhada dessa correlação.

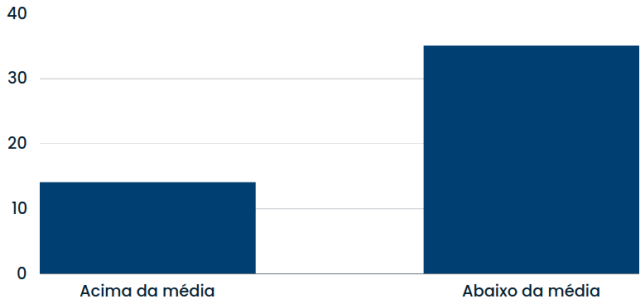

FIGURA 9. Relação entre projetos com padrão bem definido e suas linhas de código.

Ao analisarmos os projetos com um padrão bem definido, constatamos que a média de contribuidores é de 3,8. Essa média se mantém próxima à dos 76 projetos aceitos, apresentando uma ligeira diferença de apenas 0,2 a menos. Esses resultados sugerem que a quantidade de contribuidores não exerce uma influência significativa na adoção de um padrão bem definido nos projetos.

3.3 Análise de Seleção

A análise dos projetos foi realizada clonando os repositórios do github e investigando-os. Para a identificação dos padrões arquiteturais desses projetos, foi feita uma investigação das características específicas que diferem um padrão arquitetural de outro no *frontend*, além da análise da documentação do próprio projeto. Portanto, foram analisadas as seguintes características :

- Análise da estrutura de pastas e arquivos: Foi investigada a estrutura de pastas e arquivos, pois eles podem fornecer algumas indicações sobre o padrão arquitetural utilizado. Ao encontrar pastas chamadas “components” ou “containers”, por exemplo, podemos ver um indicativo de um estilo de arquitetura baseado em componentes.
- Observação do fluxo de dados : Foi observado como os dados fluem dentro do projeto, analisando se seguem a premissa de determinados padrões. Ao encontrar um projeto com uma clara separação entre a manipulação dos dados (*model*) e a apresentação dos dados (*view*) por exemplo, entendemos que isso nos leva a padrões que fazem essa separação, como o MVC e MVVM.

4. RESULTADOS

Esta seção expõe os resultados da comparação dos 76 projetos investigados. Avaliar a modularidade, coesão e acoplamento dos componentes é uma das principais preocupações da arquitetura de *software*. Por isso, investigar o padrão arquitetural usado em um projeto é um desafio, levando em consideração as variações e personalizações de um padrão, a complexidade do código e a evolução do projeto.

No estudo, diversos projetos passaram pela etapa de seleção, porém, nem todos possuem um padrão definido. Ou seja, alguns sofrem variações como as mencionadas anteriormente. Com isto, dentre os 76 projetos selecionados na etapa de seleção de estudos primários, 36,7% (28 projetos) foram classificados como "padrão indefinido", isto é, sofrem variações dentro de algum padrão. A Figura 10 apresenta uma representação visual que permite uma análise mais detalhada da relação entre os projetos que seguem um padrão bem definido e os projetos que não seguem.

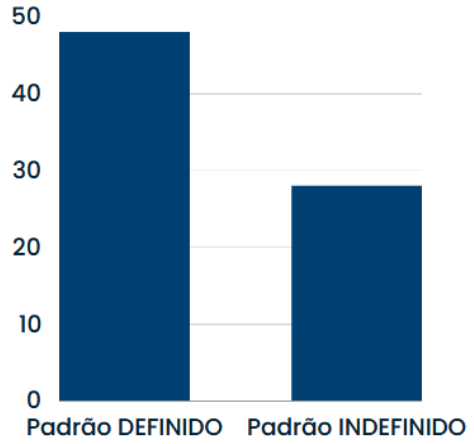

FIGURA 10. Relação entre projetos bem definidos e não definidos.

Essa porcentagem de padrões indefinidos é ainda maior quando analisamos apenas os projetos que utilizam React, sendo por volta

de 43.4%. Já quando reduzimos o escopo apenas à Angular, a porcentagem cai para 26.6%. Para uma visualização mais detalhada da relação entre os projetos que seguem um padrão bem definido e os que não seguem usando React e Angular, consulte a Figura 11.

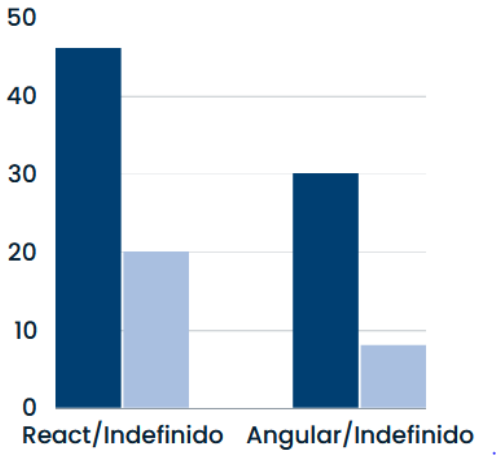

FIGURA 11. Relação entre projetos bem definidos e não definidos separados por framework.

Ao investigar os projetos que adotam o React como ferramenta de desenvolvimento e que possuem um padrão definido, é possível observar que o Flux se sobressai um pouco diante dos demais e isso ocorre pelo fato de que todos os projetos que utilizam esse padrão também utilizam o Redux, uma biblioteca de gerenciamento de estado para aplicações JavaScript que ajuda a manter a complexidade sob controle e torna o desenvolvimento mais previsível e escalável. Portanto, o Redux induz os membros da equipe a utilizarem o padrão arquitetural Flux. Podemos observar essa relação entre os projetos React na Figura 12.

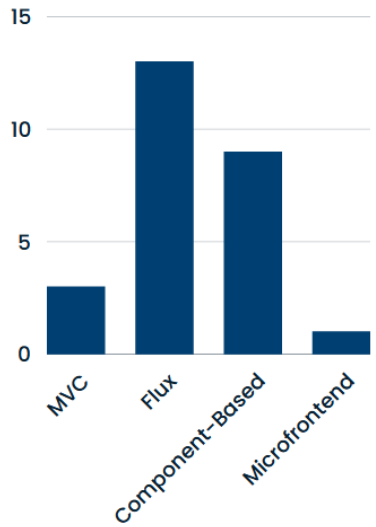

FIGURA 12. Relação entre projetos que utilizam React e possuem um padrão definido.

Já em projetos que utilizam Angular como ferramenta de desenvolvimento, ocorre algo semelhante ao Flux com React. Porém, nesse caso o padrão em questão é o MVVM. O mesmo se sobressai diante dos demais justamente por causa do próprio Angular, que favorece o uso do MVVM, fornecendo um conjunto de recursos e convenções que facilitam a implementação desse padrão. Para uma visualização mais detalhada da relação entre os padrões arquiteturais em projetos que utilizam Angular, consulte a Figura 13.

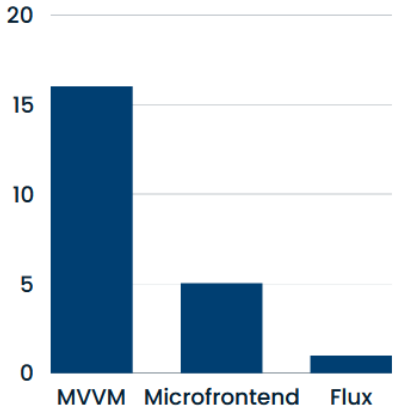

FIGURA 13. Relação entre projetos que utilizam Angular e possuem um padrão definido.

Como foi observado anteriormente, o padrão Flux foi o mais utilizado entre os projetos que utilizam React, assim como o MVVM foi o mais utilizado em projetos que utilizam Angular. Embora alguns projetos possam sofrer variações, eles geralmente são implementados de acordo com o padrão arquitetural escolhido. A utilização desses padrões é motivada principalmente pela integração com bibliotecas específicas, as quais tendem a orientar os desenvolvedores na adoção de um determinado padrão arquitetural.

5. CONCLUSÃO E TRABALHOS FUTUROS

Devido à ampla disponibilidade de bibliotecas para utilização de React e Angular, é comum a ocorrência de variações nos padrões arquiteturais. Desenvolvedores combinam frequentemente diferentes formas de organização de projetos, o que pode torná-los mais complexos para compreensão e gerenciamento à medida que evoluem. Observa-se uma preferência dos desenvolvedores em utilizar JavaScript com React, enquanto o TypeScript é frequentemente associado ao Angular. Isso explica por que grande parte dos projetos que utilizam TypeScript tendem a adotar o padrão MVVM de forma mais clara e definida.

O objetivo principal desta pesquisa foi analisar os padrões arquiteturais mais comuns em projetos que empregam as mais recentes ferramentas de desenvolvimento *frontend* do mercado [11]. Assim, este estudo buscou responder quais padrões arquiteturais são mais utilizados em projetos que adotam React e Angular como *frameworks*. Com base na análise dos resultados, pode-se inferir que nos projetos que utilizaram React, os padrões arquiteturais mais prevalentes foram Flux e Component-Based, com Flux sendo o mais utilizado. Já nos projetos que empregaram Angular, os padrões arquiteturais mais comuns foram MVVM e Micro Frontend, sendo MVVM o mais adotado. O motivo pelo

qual Flux e MVVM são os padrões arquiteturais mais utilizados em projetos React e Angular respectivamente, é justamente o fato de que o Redux e Angular facilitam a implementação desses padrões.

O Redux possui um fluxo unidirecional de dados, store centralizada e reducers puros, deixando a aplicação praticamente pronta para ser implementada usando o padrão Flux. O mesmo acontece com o Angular, que possui uma abordagem de desenvolvimento baseada em componentes e mecanismo de injeção de dependência. Esses recursos do Angular permitem separar claramente a lógica da interface do usuário e o estado da aplicação, facilitando a criação de ViewModels para manipular a interação entre a View e o Model, seguindo o padrão MVVM. Esses projetos, apesar de sofrerem algumas variações, são implementados da forma correta.

O padrão arquitetural Flux foi aplicado em um total de 14 repositórios, dos quais 13 utilizaram o React em conjunto com a biblioteca Redux e a linguagem Javascript, enquanto apenas 1 repositório optou pelo Angular. Por outro lado, o padrão arquitetural MVVM foi empregado em 16 projetos, todos eles adotando o Angular como *framework* e Typescript como linguagem.

Dos 14 projetos que adotaram o padrão arquitetural Flux, foi observado que 3 deles passaram por variações, principalmente na organização de pastas e nomenclatura de arquivos. Por outro lado, nos 16 projetos que seguiram o padrão arquitetural MVVM, todos estavam bem definidos, sem variações notáveis. Essa consistência reforça ainda mais a ideia de que o Angular facilita a implementação do MVVM, proporcionando uma estrutura clara e coerente para os projetos adotarem esse padrão arquitetural.

Além disso, buscava-se compreender as variações existentes, seus propósitos e as tendências relacionadas às linguagens e *frameworks* utilizados. No entanto, alcançar os benefícios de uma arquitetura no *frontend* tem se mostrado desafiador na prática, uma vez que cada projeto possui suas próprias especificidades, levando os desenvolvedores, por algumas vezes, a se afastarem do padrão arquitetural inicialmente planejado.

Outro grande desafio nos projetos reais é garantir a organização da equipe. É importante ressaltar que muitos projetos apresentam variações e personalizações dentro dos padrões adotados. É necessário garantir que todos membros da equipe estejam alinhados com o padrão escolhido e não se distanciem das diretrizes estabelecidas. Embora os *frameworks* sejam de grande relevância, é importante utilizá-los corretamente. A utilização dos padrões arquiteturais é motivada, em grande parte, pela integração com bibliotecas específicas que direcionam os desenvolvedores a adotar determinados padrões. A evolução do projeto também influencia na escolha e aplicação desses padrões. Portanto, os desafios técnicos e organizacionais devem ser superados para alcançar uma arquitetura bem definida em um projeto.

Em suma, esta investigação trouxe perspectivas significativas sobre os padrões arquiteturais amplamente empregados em projetos envolvendo React e Angular. Embora haja variações e desafios na definição de um único padrão em certos casos, as abordagens adotadas contribuem para a modularidade, a coesão e o acoplamento adequado dos componentes do sistema.

Como direcionamento para trabalhos futuros, é recomendada uma investigação mais aprofundada em projetos específicos, considerando aspectos relevantes, tais como desempenho, escalabilidade e manutenibilidade. Além disso, a pesquisa pode ser ampliada para abranger outras ferramentas de desenvolvimento e explorar as tendências emergentes na arquitetura de *software* para projetos de *frontend*. Dessa forma, será possível obter uma compreensão mais abrangente das práticas e abordagens arquiteturais utilizadas nesse domínio. Uma sugestão adicional para trabalhos futuros é a realização de um estudo utilizando um formulário, por meio do qual seriam levantados questionamentos junto a profissionais com vários anos de experiência na área de *frontend*. Essas perguntas poderiam abordar tópicos como os padrões arquiteturais que eles já utilizaram em projetos reais, o processo de seleção de um padrão, o impacto dos *frameworks* dentro de um padrão arquitetural, etc.

6. REFERÊNCIAS BIBLIOGRÁFICAS

[1] WILKINS, Jessica. Front-End Developer: What is Front-End Development, Explained in Plain English, 2021. Disponível em: https://www.freecodecamp.org/news/front-end-developer-what-is-front-end-development-explained-in-plain-english/. Acesso em: 16 de agosto de 2022.

[2] Code Magazine. "A Deep Dive into the Back-End for Front-End Pattern." Disponível em: https://www.codemag.com/Article/2203081/A-Deep-Dive-into-the-Back-End-for-Front-End-Pattern. Acesso em: 16 de agosto de 2022.

[3] BALDISSERA, Olívia. Quais são os tipos de arquitetura de software e como escolher o melhor para seu projeto, 2021. Disponível em: https://posdigital.pucpr.br/blog/tipos-de-arquitetura-de software. Acesso em: 19 de agosto de 2022.

[4] MONNIKENDAM, Markus. What Are Frontend and Backend in App Development?, 2021. Disponível em: https://lizard.global/blog/what-are-frontend-and-backend-in app-development. Acesso em: 19 de agosto de 2022.

[5] HERNANDEZ, Rafael. The Model View Controller Pattern - MVC Architecture and Frameworks Explained, 2021. Disponível em: https://www.freecodecamp.org/news/the-model-view explained/. Acesso em: 19 de agosto de 2022.

[6] PDVend Engineering. Let's talk about React and MVP, 2017. Disponível em: https://medium.com/pdvend-engineering/lets-talk-about-react-and-mvp-67ae35b8968c. Acesso em: 20 de agosto de 2022.

[7] OSMANI, Addy. Learning JavaScript Design Patterns, 2012. Disponível em: https://www.oreilly.com/library/view/learning-javascript-design/9781449334840/ch10s06.html. Acesso em: 12 de março de 2023.

[8] JACKSON, Cam. Micro Frontends. Martin Fowler, 2019. Disponível em: https://martinfowler.com/articles/micro frontends.html. Acesso em: 12 de março de 2023.

[9] DIMANTHA, Nipun. Flux Pattern Architecture in React. Disponível em:

https://medium.com/weekly-webtips/flux-pattern-architecture-in-react-35d0b55313f6. Acesso em: 13 de abril de 2023.

[10] NANDANIYA, Hamir. A Guide to Component-Based Architecture: Features, Benefits and more. Disponível em: https://marutitech.com/guide-to-component-based-architecture/. Acesso em: 9 de maio de 2023

[11] EBAC. Os 7 frameworks front-end mais populares. Disponível em: https://ebaconline.com.br/blog/os7-frameworks-front-end-mais-populares. Acesso em: 19 de maio de 2023