UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

LUIZ ANTONIO SCHONS

ANÁLISE COMPARATIVA DE ARQUITETURAS DE SOFTWARE PARA APLICAÇÕES WEB: DESEMPENHO E ESCALABILIDADE E EXPERIÊNCIA DE DESENVOLVIMENTO

GUARAPUAVA

2024

LUIZ ANTONIO SCHONS

ANÁLISE COMPARATIVA DE ARQUITETURAS DE SOFTWARE PARA APLICAÇÕES WEB: DESEMPENHO E ESCALABILIDADE E EXPERIÊNCIA DE DESENVOLVIMENTO

Comparative Analysis of Software Architectures for Web Applications: Performance, Scalability, and Development Experience

Projeto de Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Sistemas para Internet do Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Andres Jesse Porfirio

GUARAPUAVA

2024

SUMÁRIO

1 INTRODUÇÃO

1.1 Considerações iniciais

Desenvolver software é uma tarefa complexa e desafiadora e, nos dias atuais, com a evolução da tecnologia e a crescente demanda por aplicações web, a escolha da arquitetura de software é um fator crítico para o sucesso de um projeto. A arquitetura de software é um conjunto de padrões e práticas que definem a estrutura de um sistema de software, incluindo os componentes, as relações entre eles e as diretrizes para o seu desenvolvimento e evolução. Existem várias arquiteturas de software disponíveis para aplicações web, cada uma com suas próprias características, vantagens e desvantagens e muitas vezes é responsabilidade do arquiteto de software escolher a arquitetura mais adequada para um determinado projeto.

Fazer a escolha certa da arquitetura de software pode ter um impacto significativo no desempenho e escalabilidade de uma aplicação web, bem como na experiência de desenvolvimento dos desenvolvedores. Uma arquitetura bem projetada pode facilitar a manutenção, a evolução e a escalabilidade de um sistema de software, enquanto uma arquitetura mal projetada pode resultar em problemas de desempenho, escalabilidade e manutenção (FORD *et al.*, 2021).

A experiência de desenvolvimento é medida pela facilidade com que os desenvolvedores podem entender, modificar e estender o código de um sistema de software. Isso está atrelado a várias pontos que são mensuráveis, como o tempo de build (tempo que o sistema leva para compilar), a velocidade de execução da pipeline de CI/CD [1], e o tempo e facilidade de deploy [2].

Outro fator importante a ser considerado é a escalabilidade de uma aplicação web, que se refere à capacidade de um sistema de software lidar com um aumento na carga de trabalho, como o número de usuários, transações ou dados. Uma arquitetura escalável é aquela que pode crescer e se adaptar às demandas de um sistema de software em evolução, sem comprometer o desempenho ou a disponibilidade.

O mais crucial é que a escolha de uma arquitetura de software deve ser baseada em performance, que é medida de diversas formas, como tempo de resposta, tempo de carregamento de páginas, uso de recursos do sistema, entre outros. A performance de uma aplicação web é um fator crítico para a satisfação do usuário e o sucesso de um projeto, e uma arquitetura de software bem projetada pode contribuir significativamente para a performance de uma aplicação web.

---

[1] CI/CD cria um pipeline automatizado, desde o código-fonte até a produção, permitindo que as equipes entreguem software com mais rapidez, frequência e confiabilidade, saiba mais em https://martinfowler.com/articles/continuousIntegration.html e https://martinfowler.com/books/continuousDelivery.html

[2] Deploy é o processo de disponibilizar um software ou aplicação para uso, movendo-o de um ambiente de desenvolvimento para um ambiente de produção acessível aos usuários., saiba mais em https://www.ibm.com/docs/en/zos/3.1.0?topic=task-deploying-software

Diante disso, a escolha da arquitetura de software certa para uma aplicação web não é uma tarefa fácil, pois envolve a consideração de vários fatores, como os requisitos do projeto, as restrições de tempo e orçamento, as habilidades da equipe de desenvolvimento e as tendências tecnológicas (RICHARDS; FORD, 2020). Além disso, as arquiteturas de software estão em constante evolução, com novas abordagens e tecnologias sendo desenvolvidas regularmente, o que torna ainda mais desafiador escolher a arquitetura certa para um projeto.

A fim de ajudar os arquitetos de software a tomar decisões informadas sobre a escolha da arquitetura de software mais adequada para seus projetos, neste trabalho é proposto analisar e comparar algumas das arquiteturas de software mais populares para aplicações web, incluindo a arquitetura monolítica, a arquitetura cliente-servidor e a arquitetura serverless, com foco no desempenho e escalabilidade e na experiência de desenvolvimento dos desenvolvedores.

A análise será feita com base em um estudo de caso, onde será usado o Open Social Care [3] como aplicação web de exemplo, explorando a implementação cada uma das arquiteturas de software em um ambiente controlado e realizando testes em cada um dos setores mencionados anteriormente.

1.2 Objetivos

O objetivo deste trabalho é analisar e comparar as arquiteturas de software monolítica, cliente-servidor e serverless para aplicações web, com foco no desempenho, escalabilidade e na experiência de desenvolvimento dos desenvolvedores.

1.2.1 Objetivo específicos

- Implementar cada uma das arquiteturas de software em um ambiente de teste;
- Realizar testes de desempenho e escalabilidade em cada uma das arquiteturas;
- Avaliar a experiência de desenvolvimento dos desenvolvedores em cada uma das arquiteturas;
- Comparar os resultados dos testes e avaliações para determinar as vantagens e desvantagens de cada arquitetura.

1.3 Justificativa

A escolha da arquitetura de software certa para uma aplicação web é um fator crítico para o sucesso de um projeto. Uma arquitetura bem projetada pode facilitar a manutenção, a evolução e a escalabilidade de um sistema de software, enquanto uma arquitetura mal projetada

[3] Sistema Open Social Care, disponível em https://github.com/open-social-care

pode resultar em problemas de desempenho, escalabilidade e manutenção. Além disso, a experiência de desenvolvimento dos desenvolvedores é um fator importante a ser considerado, pois pode afetar a produtividade e a satisfação da equipe de desenvolvimento. Esse trabalho tem como objetivo ajudar os arquitetos de software a tomar decisões informadas sobre a escolha da arquitetura de software mais adequada para seus projetos, com base em testes de desempenho, escalabilidade e avaliações da experiência de desenvolvimento dos desenvolvedores.

2 TRABALHOS RELACIONADOS

Diversos estudos analisam e comparam diferentes arquiteturas de software para aplicações web. Por exemplo, Gos e Zabierowski (2020) apresentam uma comparação entre arquiteturas de microserviços e monolítica, destacando aspectos como escalabilidade, manutenção e desempenho. Já Mosleh, Dalili e Heydari (2016) propõem um framework para decisão arquitetural, auxiliando na escolha entre arquiteturas distribuídas e monolíticas com base em critérios específicos.

Felisberto (2024) discute os trade-offs entre arquiteturas monolítica e distribuída, fornecendo insights sobre as vantagens e desvantagens de cada abordagem. Além disso, Jiang, Pei e Zhao (2020) oferecem uma visão geral da arquitetura serverless, comparando-a com microserviços e destacando suas implicações organizacionais.

O trabalho proposto neste projeto busca contribuir para a área de arquitetura de software, analisando e comparando as arquiteturas monolítica, cliente-servidor e serverless para aplicações web, com foco no desempenho, escalabilidade e experiência de desenvolvimento dos desenvolvedores. A partir da análise dos resultados, espera-se identificar os principais benefícios e desafios de cada abordagem, fornecendo insights sobre como o mercado está se comportando em relação à adoção dessas arquiteturas e como as ferramentas de benchmarking podem auxiliar na escolha da melhor solução.

3 FUNDAMENTAÇÃO TEÓRICA

Para melhor compreensão do experimento realizado, é necessário entender alguns conceitos básicos sobre backend e frontend, arquitetura de software, experiência de desenvolvimento e testes de performance e análise de custo.

Nesse capítulo, serão abordados os conceitos básicos sobre cada um desses tópicos, a fim de fornecer uma base teórica para a compreensão do experimento realizado.

3.1 Divisão de camadas

O desenvolvimento de aplicações web é dividido em duas partes principais: o frontend e o backend. O frontend é a parte da aplicação que interage diretamente com o usuário, ou seja, é a interface gráfica que o usuário vê e interage. Já o backend é a parte da aplicação que fica "por trás"do frontend, ou seja, é responsável por processar as requisições do usuário, acessar o banco de dados, realizar cálculos e retornar os resultados para o frontend.

3.1.1 Frontend

O frontend é a parte da aplicação que é executada no navegador do usuário. Como está descrito na documentação MDNFrontend (2024), ele é responsável por exibir a interface gráfica da aplicação e permitir que o usuário interaja com ela. O frontend é desenvolvido utilizando tecnologias como HTML, CSS e JavaScript, e frameworks como React, Vue.js e Angular. Para desenvolver um frontend eficiente e responsivo, é importante considerar a usabilidade, a acessibilidade e a performance da aplicação.

3.1.2 Backend

Conforme a documentação do MDNBackend (2024), o backend é a parte da aplicação que é executada no servidor. Ele é responsável por processar as requisições do usuário, acessar o banco de dados, realizar cálculos e retornar os resultados para o frontend. O backend é desenvolvido utilizando tecnologias como PHP, Python, Ruby, Java e Node.js, e frameworks como Laravel, Django, Ruby on Rails, Spring e Express. Para desenvolver um backend eficiente e escalável, é importante considerar a segurança e a performance da aplicação.

Geralmente é no backend que são implementadas as regras de negócio da aplicação, como validações, autenticações, autorizações, entre outras. O backend também é responsável por garantir a integridade e a consistência dos dados da aplicação.

3.2 Arquitetura de Software

Segundo Martin (2018), a arquitetura de software é uma parte fundamental do desenvolvimento de aplicações web, pois define a estrutura e a organização do sistema, influenciando diretamente a qualidade e a manutenibilidade do código.

Existem várias arquiteturas de software disponíveis para aplicações web, cada uma com suas vantagens e desvantagens. As principais arquiteturas de software abordadas neste projeto são:

- **Monolítica**: é uma arquitetura de software em que todos os componentes da aplicação são desenvolvidos e implantados juntos. Nesse modelo, o frontend e o backend são integrados em um único sistema, o que facilita o desenvolvimento e a manutenção da aplicação, mas pode dificultar a escalabilidade e a evolução do sistema.
- **Cliente-Servidor**: é uma arquitetura de software em que a aplicação é dividida em duas partes principais: o frontend, que é executado no navegador do usuário, e o backend, que é executado no servidor. Nesse modelo, o frontend e o backend são separados, o que permite uma maior flexibilidade e escalabilidade da aplicação.
- **Serverless**: é uma arquitetura de software em que a infraestrutura de servidores é gerenciada pelo provedor de cloud, e os desenvolvedores se concentram apenas na lógica de negócio da aplicação. Nesse modelo, os serviços são executados em resposta a eventos, o que permite uma maior escalabilidade e otimização de custos.

3.2.1 Monolítica

De acordo com Fowler (2015), na arquitetura monolítica todo o código-fonte de uma aplicação web é desenvolvido e implantado como um único monólito, que consiste em vários módulos ou componentes que são interdependentes e compartilham o mesmo repositório/pasta no servidor. A arquitetura monolítica é caracterizada por ser tudo em um lugar só, tanto a parte viusal quanto a parte lógica. A Figura 1 ilustra essa arquitetura.

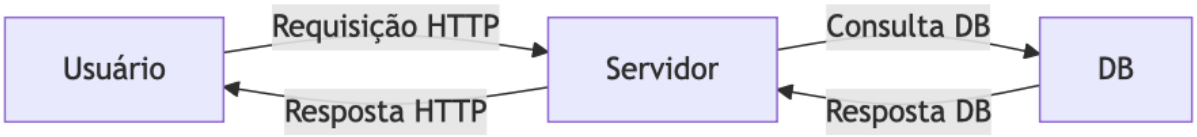

Figura 1 – Arquitetura Monolitica

A arquitetura monolítica é composta por três camadas principais: a camada de apresentação, a camada de lógica de negócios e a camada de acesso a dados. A camada de apresentação é responsável por interagir com os usuários, exibindo a interface do usuário e coletando

entradas do usuário. A camada de lógica de negócios é responsável por implementar a lógica de negócios da aplicação web, processando as entradas do usuário e gerando as saídas correspondentes. A camada de acesso a dados é responsável por interagir com o banco de dados, recuperando e armazenando dados para a aplicação web.

Essa divisão de camadas não é uma regra, mas sim uma prática comum na arquitetura monolítica, que pode variar dependendo das necessidades e requisitos de um projeto específico. A arquitetura monolítica é caracterizada por sua simplicidade, facilidade de desenvolvimento e implantação, e baixa complexidade, o que a torna uma escolha popular para pequenas e médias aplicações web.

3.2.2 Cliente-Servidor

Nesta arquitetura, o sistema de software é dividido em duas partes principais: o cliente, que é responsável por interagir com os usuários e exibir a interface do usuário, e o servidor, que é responsável por processar as solicitações dos clientes, executar a lógica de negócios e acessar os dados. A Figura 2 ilustra essa arquitetura.

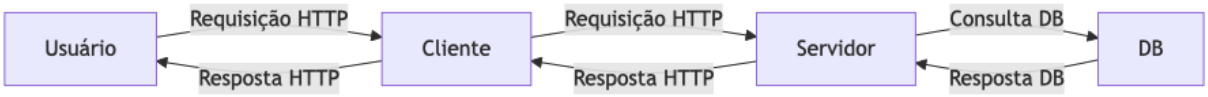

Figura 2 – Arquitetura Cliente Servidor

Segundo Sommerville (2018), a arquitetura cliente-servidor é caracterizada pela divisão de responsabilidades entre o cliente e o servidor, permitindo uma comunicação eficiente e estruturada entre as partes.

A arquitetura cliente-servidor é composta por duas camadas principais: a camada do cliente e a camada do servidor. A camada do cliente é responsável por interagir com os usuários, exibindo a interface do usuário e coletando entradas do usuário. A camada do servidor é responsável por processar as solicitações dos clientes, executar a lógica de negócios e acessar os dados.

3.2.3 Serverless

Segundo Fowler (2018), na arquitetura serverless o sistema de software é dividido em várias funções independentes e autônomas, que são executadas em um ambiente de execução gerenciado por um provedor de serviços de nuvem. Cada função é responsável por executar uma tarefa específica, como processar uma solicitação HTTP, acessar um banco de dados ou enviar um e-mail. A Figura 3 ilustra essa arquitetura, onde uma API Gateway é uma camada intermediária que recebe as solicitações HTTP dos usuários e as encaminha para as funções

apropriadas. Cada função é executada em um ambiente de execução isolado e é dimensionada automaticamente de acordo com a demanda.

Figura 3 – Arquitetura Serverless

A arquitetura Serverless está muito relacionada ao ambiente de Cloud [1], segundo Fowler (2018).

3.3 Experiência de Desenvolvimento

No artigo Fagerholm e Münch (2012), os autores Fabian Fagerholm e Jürgen Münch definem a experiência de desenvolvimento como "a percepção do desenvolvedor sobre a qualidade do ambiente de desenvolvimento e a facilidade de uso das ferramentas e práticas de desenvolvimento". Em outras palavras, a experiência de desenvolvimento está diretamente relacionada à eficiência, satisfação e produtividade dos desenvolvedores ao trabalhar em um projeto.

Uma boa experiência de desenvolvimento envolve fatores como facilidade na configuração do ambiente, simplicidade na execução de builds e testes, automação de processos, documentação clara e disponibilidade de ferramentas eficazes. Arquiteturas que exigem configurações complexas ou processos manuais extensivos tendem a impactar negativamente a experiência dos desenvolvedores, tornando o desenvolvimento mais demorado e propenso a erros.

Para avaliar a experiência de desenvolvimento das diferentes arquiteturas testadas, foram analisados aspectos como a quantidade de passos necessários para realizar um deploy, o tempo total para rodar a suíte de testes automatizados e a facilidade de configuração do ambiente de desenvolvimento. No caso da arquitetura monolítica, por exemplo, o processo de deploy envolve **X** passos, enquanto na arquitetura cliente-servidor, esse número sobe para **Y** devido à necessidade de configurar múltiplos serviços separadamente. Além disso, o tempo médio para executar todos os testes foi de aproximadamente **Z** minutos na arquitetura monolítica, comparado a **W** minutos na arquitetura cliente-servidor e **V** minutos na arquitetura serverless. Esses fatores serão analisados para determinar qual abordagem proporciona uma melhor experiência de desenvolvimento.

[1] A "nuvem"(cloud) refere-se à entrega sob demanda de recursos de computação—como servidores, armazenamento, bancos de dados, redes, software, análises e inteligência—pela Internet ("a nuvem"). As empresas que oferecem esses recursos de computação são chamadas de provedores de nuvem e normalmente cobram pelos serviços de nuvem com base no uso, semelhante à forma como você pagaria pela sua conta de água ou luz em casa.

3.4 Análise de Custo e Testes de Performance

Metrificar a performance de uma aplicação em diferentes arquiteturas não é uma tarefa simples e requer uma análise criteriosa. Para isso, é necessário realizar testes de performance e análise de custo, a fim de identificar gargalos de desempenho, otimizar o uso de recursos e reduzir os custos operacionais.

Uma forma geral de realizar comparativos é com custos, pois isso é universal e pode ser aplicado em qualquer cenário. O autor Lorraine S. Lee e R. David Mautz Jr, no artigo Lee e Jr (2012), aborda a importância de gerenciar os custos de infraestrutura em ambientes de *cloud computing*, e como isso pode impactar diretamente a eficiência operacional e financeira de uma aplicação.

Com base nesses conceitos, é possível realizar uma análise de custo para comparar as arquiteturas monolítica, baseada em microserviços e serverless, a fim de identificar as vantagens e desvantagens de cada abordagem.

Outra forma de realizar comparativos é com testes de performance, pois isso é fundamental para garantir a qualidade e a escalabilidade de uma aplicação. A ferramenta PEST Maduro (2019), criada pelo Nuno Maduro, é uma ferramenta de teste de performance para aplicações PHP, que permite medir o tempo de execução de uma requisição e identificar possíveis gargalos de desempenho.

Com base nesses conceitos, é possível realizar testes de performance para comparar as arquiteturas monolítica, baseada em microserviços e serverless, a fim de identificar as vantagens e desvantagens de cada abordagem.

4 METODOLOGIA

A metodologia adotada para a realização deste trabalho consiste nas seguintes etapas:

Migração do Open Social Care para a arquitetura monolítica: Realizar uma cópia do código-fonte original do sistema Open Social Care, anteriormente implementado como uma arquitetura cliente-servidor;

Configurar um ambiente de desenvolvimento para rodar o projeto em um único container Docker capaz de prover tanto os recursos de backend quanto os de frontend a partir de um único monólito;

Migrar e ajustar o frontend do sistema, anteriormente isolado do backend, para que se integre ao backend como um único monólito.

Definição dos critérios e métricas para a análise comparativa de arquiteturas de software: Esta etapa envolve a definição de critérios e métricas para avaliar o desempenho, a escalabilidade e a experiência de desenvolvimento das arquiteturas monolítica, cliente-servidor e serverless. Os critérios incluem tempo de resposta, tempo de carregamento de páginas, uso de recursos do sistema, tempo de build, velocidade de execução da pipeline de CI/CD, facilidade de deploy e custos de infraestrutura.

Elaboração de cenários experimentais: Criar cenários experimentais para testar as arquiteturas monolítica, cliente-servidor e serverless em diferentes condições de carga e uso. Os cenários incluem testes de desempenho, escalabilidade e experiência de desenvolvimento, com base nos critérios e métricas definidos na etapa anterior.

Aplicação das métricas e realização das análises nos cenários experimentais: Realizar testes de desempenho e escalabilidade em cada uma das arquiteturas, com base nos cenários experimentais definidos. Os resultados serão analisados e comparados para identificar as vantagens e desvantagens de cada arquitetura em relação aos critérios e métricas estabelecidos. Dessa forma, será possível avaliar a performance e a escalabilidade de cada arquitetura, bem como a experiência de desenvolvimento dos desenvolvedores.

Documentação e discussão dos resultados: Com os resultados dos testes e análises em mãos, fazer a documentação e discussão dos resultados, com base nos critérios e métricas definidos. Serão identificadas as principais vantagens e desvantagens de cada arquitetura, a fim de auxiliar os arquitetos de software na escolha da arquitetura mais adequada para seus projetos.

5 CRITÉRIOS E MÉTRICAS PARA O EXPERIMENTO

Para avaliar as arquiteturas monolítica, cliente-servidor e serverless, serão utilizados os seguintes critérios e métricas:

1. **Uso de recursos do sistema:**
   Esta métrica visa avaliar o consumo de recursos do sistema, como CPU, memória e disco, em cada uma das arquiteturas, a fim de identificar possíveis gargalos de desempenho e otimizar o uso de recursos.

2. **Tempo de resposta:**
   Será avaliado o tempo de resposta de cada arquitetura em diferentes condições de carga e uso, a fim de identificar a capacidade de cada arquitetura de lidar com um aumento na carga de trabalho.

3. **Tempo de carregamento de páginas:**
   Será avaliado o tempo de carregamento de páginas em cada uma das arquiteturas, com o objetivo de comparar cada uma delas e entender como a arquitetura impacta na experiência do usuário.

4. **Tempo de build:**
   Será avaliado o tempo que o sistema leva para compilar em cada uma das arquiteturas. Essa métrica é importante para a experiência do desenvolvedor, pois quanto mais rápido o build, mais produtivo o desenvolvedor será.

5. **Velocidade de execução da pipeline de CI/CD:**
   Será avaliada a velocidade de execução da pipeline de CI/CD em cada uma das arquiteturas. Essa métrica é importante para a eficiência do processo de desenvolvimento, pois quanto mais rápido a pipeline, mais rápido o desenvolvedor terá feedback sobre o código.

6. **Tempo e facilidade de deploy:**
   Será avaliado o tempo e a facilidade de deploy (quantidade de passos, complexidade, etc.) em cada uma das arquiteturas, para entender como a arquitetura impacta no processo de deploy, na produtividade e na eficiência do desenvolvedor.

7. **Custos de infraestrutura:**
   Será avaliado o custo de infraestrutura em cada uma das arquiteturas, a fim de identificar a relação custo-benefício de cada uma delas e entender como a arquitetura impacta nos custos operacionais da aplicação.

Com base nessas métricas, será possível comparar as arquiteturas monolítica, cliente-servidor e serverless afim de identificar as vantagens e desvantagens de cada uma delas em relação aos critérios estabelecidos.

6 CENÁRIO EXPERIMENTAL

Para elaborar os experimentos necessários, foi selecionada a aplicação Open Social Care, que já passou por diferentes versões contemplando parte das arquiteturas que serão utilizadas no experimento. O Open Social Care foi inicialmente implementado em uma arquitetura serverless, utilizando o Firebase, e posteriormente migrado para uma arquitetura cliente-servidor, com o frontend em React e o backend em Laravel. No entanto, a arquitetura monolítica ainda não havia sido explorada no sistema Open Social Care, e por isso foi considerado importante ter essa versão também.

6.1 Open Social Care

O Open Social Care é um sistema para gerenciamento de atendimentos sociais, criado para auxiliar o dia a dia de trabalho de assistentes sociais do CCG (Conselho da Comunidade de Guarapuava). O sistema surgiu como um projeto de disciplina de Sistemas Distribuídos da UTFPR, câmpus Guarapuava, quando uma assistente social do CCG entrou em contato com o professor da disciplina e sugeriu a ideia de informatização do processo até então utilizado no CCG. Através de reuniões e discussões, foi feita a coleta dos requisitos e planejado como o sistema teria que funcionar. Por fim, foi desenvolvido um protótipo como um objeto de estudos na disciplina. Com o passar do tempo, houve a necessidade de uma evolução no projeto, pois ele iniciou sendo Serverless usando o Firebase e com o tempo foi migrado para outras tecnologias e para a arquitetura Cliente-Servidor.

O projeto Open Social Care foi abordado em estudos anteriores, como o (SOUZA; PORFIRIO, 2024), que detalha a transição da arquitetura serverless para a cliente-servidor no backend, e o (BOEIRA; PORFIRIO, 2024), que discute a implementação do frontend.

6.2 Cenário para a Arquitetura Monolítica

Para ter uma versão monolítica do Open Social Care, será necessário integrar o frontend e o backend em um único projeto, garantindo que todas as funcionalidades do sistema estejam funcionando corretamente. O frontend, que atualmente é um projeto React, precisará ser integrado ao backend, que atualmente é um projeto Laravel, em um único projeto monolítico. Dessa forma, será possível avaliar o desempenho, a escalabilidade e a experiência de desenvolvimento da arquitetura monolítica em comparação com as arquiteturas cliente-servidor e serverless.

7 RESULTADOS PRELIMINARES

7.1 Elaboração dos cenários experimentais

7.1.1 Arquitetura serverless

Para a arquitetura serverless, foi aproveitado o deploy existente do Open Social Care, que utiliza o Firebase como plataforma serverless. O frontend é um projeto React hospedado no serviço Firebase Hosting[1], e o backend é um projeto Firebase Functions que responde às requisições do frontend. Dessa forma, foi possível manter a arquitetura serverless do Open Social Care para os experimentos.

7.1.2 Arquitetura cliente-servidor

Para a arquitetura cliente-servidor, foi gerado um release no GitHub da versão 2 do Open Social Care, que foi desenvolvida em (BOEIRA; PORFIRIO, 2024) e (SOUZA; PORFIRIO, 2024), e que utiliza uma arquitetura cliente-servidor, com o frontend em Next.js e o backend em Laravel. O frontend foi hospedado em uma VPS, e o backend foi implantado em um servidor separado, garantindo a separação entre o frontend e o backend. Dessa forma, foi possível manter a arquitetura cliente-servidor do Open Social Care para os experimentos.

Uma arquitetura cliente-servidor é uma arquitetura de software em que a aplicação é dividida em duas partes principais: o frontend, que é executado no navegador do usuário, e o backend, que é executado no servidor. Nesse modelo, o frontend e o backend são separados, o que permite uma maior flexibilidade e escalabilidade da aplicação, conforme explica o artigo (GOS; ZABIEROWSKI, 2020).

7.1.3 Arquitetura monolítica

Para a arquitetura monolítica, foi necessário integrar o frontend e o backend do Open Social Care em um único projeto, garantindo que todas as funcionalidades do sistema estejam funcionando corretamente.

O projeto frontend ainda continua sendo um projeto React, mas eles rodam no mesmo projeto Laravel, o que facilita a integração entre o frontend e o backend. Dessa forma, foi possível avaliar o desempenho, a escalabilidade e a experiência de desenvolvimento da arquitetura monolítica em comparação com as arquiteturas cliente-servidor e serverless.

Dentro da pasta ‘resources/js‘ do Laravel, foi iniciado um novo projeto React, e o código do frontend foi migrado para dentro desse projeto. Dessa forma, o frontend e o backend do

[1] saiba mais em https://firebase.google.com/docs/hosting?hl=pt

Open Social Care estão integrados em um único projeto monolítico, que será utilizado para os experimentos.

7.2 Discussão dos resultados

Até o momento, foi possível elaborar os cenários experimentais para as arquiteturas serverless, cliente-servidor e monolítica do Open Social Care. A arquitetura serverless foi mantida com o Firebase, a arquitetura cliente-servidor será implantada em uma VPS e a arquitetura monolítica foi integrada em um único projeto Laravel que também será implantado em uma VPS.

Os próximos passos incluem a realização dos testes de desempenho, escalabilidade e experiência de desenvolvimento em cada uma das arquiteturas, com base nos critérios e métricas definidos. Com os resultados dos testes em mãos, será possível comparar as arquiteturas e identificar as vantagens e desvantagens de cada uma delas em relação aos critérios estabelecidos.

Alguns desafios enfrentados até o momento incluem a integração do frontend e do backend em um único projeto monolítico, a configuração do ambiente de desenvolvimento para rodar o projeto em um container Docker e a implantação dos projetos em uma VPS. No entanto, esses desafios foram superados com sucesso, e os experimentos estão prontos para serem realizados.

Além disso, a migração do sistema para a arquitetura monolítica foi concluída com êxito. Durante esse processo, foram enfrentadas dificuldades relacionadas ao roteamento e à autenticação, uma vez que o ambiente em que a aplicação foi implantada estava originalmente preparado apenas para receber o backend. Dessa forma, ajustes foram necessários para que o ambiente suportasse corretamente a execução da aplicação completa, garantindo seu pleno funcionamento.

8 CONCLUSÃO

A escolha da arquitetura de software desempenha um papel fundamental no sucesso de projetos de desenvolvimento de aplicações web, influenciando diretamente o desempenho, a escalabilidade e a experiência dos desenvolvedores. Este estudo teve como objetivo comparar as arquiteturas monolítica, cliente-servidor e serverless, focando nesses aspectos cruciais.

A análise dos resultados obtidos nos testes de desempenho, escalabilidade e experiência de desenvolvimento permitirão identificar as vantagens e desvantagens de cada abordagem. Esses insights são essenciais para orientar arquitetos de software na seleção da arquitetura mais adequada, considerando as necessidades específicas de cada projeto.

Além de fornecer métricas detalhadas sobre o desempenho e a escalabilidade de cada arquitetura, este trabalho também pretende analisar as implicações da migração entre elas, exemplificada pela experiência do Open Social Care. Essa análise destacou os principais benefícios e desafios associados a cada abordagem, oferecendo uma compreensão mais profunda do impacto dessas arquiteturas no contexto de desenvolvimento.

Com base nos resultados obtidos, espera-se que este estudo contribua para a tomada de decisões informadas na escolha da arquitetura mais apropriada para diferentes projetos, fornecendo uma visão abrangente sobre as tendências atuais na adoção dessas arquiteturas e o papel das ferramentas de benchmarking na seleção da solução mais eficaz.

REFERÊNCIAS

BOEIRA, S. L. dos S.; PORFIRIO, A. J. **Implementação e Atualização de Frontend para o Sistema Open Social Care**. 2024. 53 f. Monografia (Trabalho de Conclusão de Curso) — Universidade Tecnológica Federal do Paraná, Guarapuava, 2024. Disponível em: https://tcc.tsi.pro.br/uploads/academic_activity/pdf/267/GP_COINT_2024_1_CAMILA_EMANUELE_DE_SOUZA_MONOGRAFIA.pdf.

FAGERHOLM, F.; MÜNCH, J. Developer experience: Concept and definition. *In*: IEEE. **2012 international conference on software and system process (ICSSP)**. [*S.l.*], 2012. p. 73–77.

FELISBERTO, M. The trade-offs between monolithic vs. distributed architectures. **arXiv preprint arXiv:2405.03619**, 2024.

FORD, N. *et al.* **Software Architecture: The Hard Parts : Modern Trade-off Analysis for Distributed Architectures**. O'Reilly, 2021. ISBN 9781492086895. Disponível em: https://books.google.com.br/books?id=sWNozgEACAAJ.

FOWLER, M. **Bliki: Monolith First**. 2015. Disponível em: https://martinfowler.com/bliki/MonolithFirst.html.

FOWLER, M. **Articles: Serverless**. 2018. Disponível em: https://martinfowler.com/articles/serverless.html.

GOS, K.; ZABIEROWSKI, W. The comparison of microservice and monolithic architecture. *In*: IEEE. **2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)**. [*S.l.*], 2020. p. 150–153.

JIANG, L.; PEI, Y.; ZHAO, J. Overview of serverless architecture research. **Journal of Physics: Conference Series**, IOP Publishing, v. 1453, n. 1, p. 012119, jan 2020. Disponível em: https://dx.doi.org/10.1088/1742-6596/1453/1/012119.

LEE, L. S.; JR, R. D. M. Using cloud computing to manage costs. **Journal of Corporate Accounting & Finance**, Wiley Online Library, v. 23, n. 3, p. 11–15, 2012.

MADURO, N. **PEST**. 2019. Disponível em: https://pestphp.com.

MARTIN, R. **Clean Architecture: A Craftsman's Guide to Software Structure and Design**. Prentice Hall, 2018. (Martin, Robert C). ISBN 9780134494166. Disponível em: https://books.google.com.br/books?id=8ngAkAEACAAJ.

MDNBACKEND. **Programação de site do lado do servidor**. 2024. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/Server-side.

MDNFRONTEND. **Desenvolvedor Web Front-end**. 2024. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/Front-end_web_developer.

MOSLEH, M.; DALILI, K.; HEYDARI, B. Distributed or monolithic? a computational architecture decision framework. **IEEE Systems journal**, IEEE, v. 12, n. 1, p. 125–136, 2016.

RICHARDS, M.; FORD, N. **Fundamentals of Software Architecture: An Engineering Approach**. O'Reilly Media, Incorporated, 2020. ISBN 9781492043454. Disponível em: https://books.google.com.br/books?id=_pNdwgEACAAJ.

SOMMERVILLE, I. **Engenharia de Software**. 10. ed. São Paulo: Pearson, 2018. Disponível em: https://www.facom.ufu.br/~william/Disciplinas%202018-2/BSI-GSI030-EngenhariaSoftware/Livro/engenhariaSoftwareSommerville.pdf.

SOUZA, C. E. D.; PORFIRIO, A. J. **Atualização do Backend do Sistema Open Social Care: Migrando da arquitetura serverless para uma API em Laravel**. 2024. 56 f. Monografia (Trabalho de Conclusão de Curso) — Universidade Tecnológica Federal do Paraná, Guarapuava, 2024. Disponível em: https://tcc.tsi.pro.br/uploads/academic_activity/pdf/267/GP_COINT_2024_1_CAMILA_EMANUELE_DE_SOUZA_MONOGRAFIA.pdf.