DESENVOLVIMENTO COM JAMSTACK

RESUMO

No cenário competitivo atual, os desenvolvedores precisam utilizar as arquiteturas, os modelos e os frameworks de desenvolvimento mais eficientes. É preciso que utilizem ferramentas capazes de auxiliá-los a desenvolver sistemas de forma mais eficaz, tanto do ponto de vista do desempenho do desenvolvimento em si quanto em algumas funcionalidades que o sistema pode ter como o SEO, por exemplo (Search Engine Optimization). É importante que artigos acadêmicos abordem este tema para que seja analisado por estudantes, profissionais e empresários que se interessam pela área. Para satisfazer esta necessidade, este trabalho tem como objetivo apresentar ao leitor os fundamentos do JAMStack, suas vantagens, seus frameworks e as tecnologias utilizadas.

Palavras-chave: JAMStack; SEO; Desenvolvimento Web

ABSTRACT

In today's competitive landscape, developers need to use the most efficient architectures, models and development frameworks. It is necessary that they use tools capable of helping them to develop systems more effectively, both from the point of view of development performance itself and in some features that the system may have, such as SEO, for example (Search Engine Optimization). It is important that academic articles address this topic so that it can be analyzed by students, professionals and entrepreneurs who are interested in the area. To satisfy this need, this work aims to present to the reader the fundamentals of JAMStack, its advantages, its frameworks and the technologies used.

Keywords: JAMStack; IF THE; Web development

SUMÁRIO

1 INTRODUÇÃO

A demanda por desenvolvimento de aplicações Web é crescente e esta demanda requer eficiência no desenvolvimento, tanto do pondo de vista das horas de dedicação requeridas pelo desenvolvedor quanto em relação as funcionalidades implementadas, como o SEO. Uma das formas que vêm sido utilizadas para se implementar estes sistemas é o JAMStack, que é uma arquitetura que vem sido utilizada para se criar sites mais rápido.

Dado a importância desta arquitetura para o mercado, é importante que seja feita uma análise aprofundada da mesma. Esta análise deve mostrar seus pontos positivos e negativos. Além disso, é importante que seja feita uma comparação com outras arquiteturas e também que os frameworks utilizados para esta arquitetura sejam apresentados no estudo para análise de profissionais, estudantes e empresários que precisem destas informações.

1.1 Objetivo

Neste sentido, o objetivo deste trabalho é apresentar uma análise ampla sobre a arquitetura do JAMStack juntamente com uma comparação de arquiteturas de implementação de sistemas web que também são utilizadas. Para isso, os seguintes objetivos específicos são apresentados ao longo deste texto:

- Apresentar a arquitetura JAMStack
- Apresentar os desafios do desenvolvimento Web atualmente e como o JAMStack pode ajudar a solucionar estes desafios.
- Apresentar alguns frameworks utilizados no desenvolvimento de sistemas com arquitetura JAMStack.

1.2 Metodologia

Este trabalho será desenvolvido com base em uma pesquisa bibliográfica exploratória de artigos relevantes. Será priorizados artigos mais

recentes publicados nos últimos 5 anos, mas alguns artigos podem ser anteriores a este período e serão utilizados devido a relevância de seu conteúdo.

2 JAMSTACK

O JAMstack reúne JavaScript, APIs e marcação, os três componentes principais usados para criar sites rápidos e altamente dinâmicos. Os sites JAMstack são adequados para atender aos exigentes requisitos da web mobile - first de hoje (onde os tempos de carregamento são urgentemente importantes e a largura de banda de qualidade nunca pode ser garantida).

Devemos fazer uma pausa para dizer o que o JAMstack não é: não é uma tecnologia específica em si mesma; nem é dirigido por uma grande empresa; nem existe um órgão de padrões que o controle ou defina (MARKOVIC, 2021).

Em vez disso, o JAMstack é um movimento, uma coleção da comunidade de melhores práticas e fluxos de trabalho que resultam em sites de alta velocidade nos quais é um prazer trabalhar (MARKOVIC, 2021).

Ao mergulhar, encontramos algumas diretrizes comuns, mas também muitas opções, incluindo sua escolha de idiomas. JavaScript é chamado especificamente como a linguagem do navegador, mas pode-se usar tanto ou tão pouco JavaScript quanto seu projeto requer. Muitos sites JAMstack também usam Python, Go ou Ruby para modelagem e lógica. Muitos softwares de código aberto surgiram para ajudá-lo a criar sites JAMstack, e passamos algum tempo neste livro analisando algumas das opções mais populares (MARKOVIC, 2021; BILLINGSLEY, 2020).

Nem tudo no JAMstack é uma ideia nova, mas só muito recentemente tivemos a tecnologia necessária para tornar a abordagem possível, especialmente na borda da rede e em navegadores. Esta primeira seção do livro deve fornecer uma compreensão prática do JAMstack, por que é um novo desenvolvimento importante e como raciocinar sobre isso (MARKOVIC, 2021; BILLINGSLEY, 2020).

2.1 Um Workflow para Produtividade e Desempenho

Por natureza, os sites JAMstack são os seguintes (MARKOVIC, 2021; BILLINGSLEY, 2020; GHOSHAL, 2008):

- Distribuído globalmente e resiliente ao tráfego pesado
- fluxo de trabalho baseado em Git amigável ao desenvolvedor
- Projetado de forma modular, consumindo outros serviços via APIs
- Pré-construído e otimizado antes de ser servido

É este último ponto que merece atenção especial. Pensemos por um momento sobre a abordagem mais comum de hoje para servir conteúdo da web: para cada solicitação feita a um site, os dados são extraídos de um banco de dados, renderizados em um modelo, processados em HTML e, finalmente, enviados pela rede (e talvez até um oceano) para o navegador que o solicitou (MARKOVIC, 2021; BILLINGSLEY, 2020; GHOSHAL, 2008).

Quando os servidores da Web constroem repetidamente cada página para cada solicitação, começa a parecer que muito trabalho acontece no lugar errado na hora errada. Afinal, uma regra geral para o máximo desempenho é realizar o menor número de etapas possível. O novo HTML não deveria ser produzido apenas quando o conteúdo ou os dados mudam, e não sempre que é solicitado? (MARKOVIC, 2021; BILLINGSLEY, 2020; GHOSHAL, 2008)

Acontece que é exatamente assim que o JAMstack opera. Veja o que acontece no fluxo de trabalho do JAMstack (MARKOVIC, 2021; BILLINGSLEY, 2020; GHOSHAL, 2008):

1. A fonte do site é um repositório hospedado que armazena conteúdo e código juntos como arquivos editáveis.
2. Sempre que uma alteração é feita, é acionado um processo de construção que pré-renderiza o site, criando o HTML final a partir de modelos, conteúdo e dados.
3. Os ativos preparados e renderizados são publicados globalmente em uma CDN, colocando-os o mais próximo possível fisicamente dos usuários finais.

Essa abordagem elimina grandes quantidades de latência de servidor e rede. Dada a eficiência e simplicidade de servir conteúdo pré -renderizado

diretamente de uma CDN, não é surpresa que os sites JAMstack tendam a obter as pontuações mais altas possíveis em testes de velocidade como o Google Lighthouse (MARKOVIC, 2021; BILLINGSLEY, 2020; GHOSHAL, 2008).

O conteúdo distribuído globalmente não é totalmente novo, mas a velocidade com que se pode atualizar arquivos CDN diretamente de um repositório é nova. Chega de atrasos temidos esperando que o cache do CDN expire – agora podemos construir sites altamente distribuídos na borda da rede e eliminar a necessidade de qualquer tipo de servidor front-end para processar cada solicitação (MARKOVIC, 2021; BILLINGSLEY, 2020; GHOSHAL, 2008).

No JAMstack, é possível e comum que o conteúdo do site, posts de blog e tudo, fique em um repositório Git ao lado do código e dos templates. Isso significa que nenhum banco de dados de conteúdo é necessário, tornando os sites JAMstack muito mais fáceis de configurar, executar, implantar, ramificar e modificar (MARKOVIC, 2021; BILLINGSLEY, 2020; GHOSHAL, 2008).

E neste mundo centrado no controle de versão do JAMstack, cada implantação do site é atômica, tornando trivial a reversão para qualquer estado, a qualquer momento, por qualquer motivo. Ambientes de teste complexos não são mais necessários, pois as alterações de visualização e teste usam o sistema de ramificação integrado ao coração do Git. Com o fluxo de trabalho para alterações feito de forma rápida, simples e segura, a maioria dos sites JAMstack está longe de ser “estático” e muitas vezes são atualizados com a mesma frequência que suas contrapartes mais complexas, às vezes centenas de vezes por dia (MARKOVIC, 2021; BILLINGSLEY, 2020; GHOSHAL, 2008).

2.2 Fundamentos

Ao imaginar um fluxo de trabalho do JAMstack, provavelmente imaginamos fazer alterações de código em um editor e, em seguida, executar um comando para construir o site e implantá-lo na produção. Um pouco de desenvolvimento no JAMstack acontece exatamente dessa maneira, especialmente no início (MARKOVIC, 2021; BILLINGSLEY, 2020).

Mas para a maioria dos sites JAMstack, os contribuidores não são apenas desenvolvedores. Novas atualizações no site também são acionadas

por autores de conteúdo usando um CMS, bem como por ações automatizadas, exatamente como se esperaria de qualquer site moderno (MARKOVIC, 2021; BILLINGSLEY, 2020).

Em vez de acontecer localmente, o processo de compilação agora geralmente é executado em um serviço hospedado na nuvem. Envie uma alteração para o GitHub (ou outro serviço de repositório) e uma nova compilação é acionada automaticamente em servidores de compilação de finalidade especial, enviando o resultado final diretamente para a CDN. É uma maneira confortável de desenvolver (MARKOVIC, 2021; BILLINGSLEY, 2020).

Mas e aqueles autores que não são desenvolvedores e podem não estar familiarizados com o Git? O JAMstack gerou uma nova geração inteligente de ferramentas de autoria que parecem e funcionam como um Content Management System (CMS) normal, mas na verdade verificam as alterações no controle de versão nos bastidores. Dessa forma, todos participam do mesmo fluxo de trabalho, aproveitando a segurança, ramificações e reversões do software de controle de versão moderno, mesmo que não saibam que isso está acontecendo. É uma boa melhoria em relação ao conteúdo que requer um banco de dados que precisamos gerenciar e versão separadamente (MARKOVIC, 2021; BILLINGSLEY, 2020).

Também é possível contribuir com alterações no site de uma terceira maneira: programaticamente, via automação. Por exemplo, é possível executar um script diariamente que incorpore os últimos artigos da imprensa e menções do Twitter na página inicial. Ou veja o que acontece no site JAMStack da Smashing Magazine: cada vez que um usuário comenta um artigo, uma função simples envia o comentário para o repositório do site e aciona uma nova compilação (desde que o comentário passe pela moderação).

Começando a ver o poder desse fluxo de trabalho? Desenvolvedores, autores de conteúdo e processos automatizados estão salvando um histórico vivo do site no mesmo lugar – o repositório – que atua como uma fonte de verdade. Mesmo que o site seja atualizado centenas de vezes em um dia ou mais, cada estado é preservado. E, o mais importante, o site permanece rápido e responsivo porque cada página é construída e otimizada antes de ser veiculada (MARKOVIC, 2021; BILLINGSLEY, 2020).

2.3 Desempenho do JAMStack Quanto Uso do SEO

O uso de JAMStack favorece o uso do SEO por conta de *(i)* as páginas estáticas serem carregadas instantaneamente e *(ii)* os motores de busca do google e similares premiarem a leitura mais fácil do conteúdo dessas páginas por parte de seus robôs (MARKOVIC, 2021; ZAMMETI, 2020).

2.4 API para Processar e Personalizar

No entanto, é preciso mais do que HTML: os aplicativos da Web precisam realizar trabalho e ter estado. Tradicionalmente, os servidores da Web eram obrigados a armazenar a sessão de um usuário e permitir que o aplicativo fizesse coisas como lembrar de itens em um carrinho de compras ou alimentar a experiência de checkout.

No JAMstack, essas experiências personalizadas são feitas usando JavaScript para fazer chamadas de API que enviam e recebem dados. Muitas das APIs usadas são serviços de terceiros. O Stripe, por exemplo, é um serviço popular para processamento de pagamentos. Algolia é uma API popular para potencializar a pesquisa. (Seus autores têm um amor profundo por quase todas as tecnologias, mas nunca voltariam a lidar manualmente com pagamentos ou executar clusters de pesquisa Apache Solr) (MARKOVIC, 2021).

Outras APIs podem ser funções personalizadas exclusivas para cada aplicativo. Em vez de uma estrutura monolítica e complexa que gerencia tudo, os sites agora podem ser projetados em torno de microsserviços : funções simples que executam tarefas específicas, executando uma vez quando chamadas e saindo de forma limpa. É uma abordagem muito escalável que é fácil de raciocinar (MARKOVIC, 2021).

Mas sem servidores para armazenar sessões de usuário, como conectamos todas essas chamadas de API discretas? Como lidamos com autenticação e identidade? Para alimentar contas de usuário, um aplicativo JAMstack geralmente cria um token de ID seguro armazenado no navegador. (Esse tipo de token é chamado de JavaScript Web Token, ou JWT.) A identidade é então passada com cada chamada de API para que os serviços estejam cientes do usuário. Abordaremos essa configuração com mais

detalhes posteriormente, mas a apresentaremos aqui para ajudá-lo a entender o poder da plataforma. Os sites JAMstack podem fazer muito mais do que fornecer conteúdo simples, e muitos são aplicativos avançados com todos os recursos que se espera, como comércio eletrônico, associação e personalização avançada (MARKOVIC, 2021).

Sites estáticos podem ser, bem, estáticos, mas descobriremos que sites JAMstack são tudo menos isso. Na verdade, veremos que a marcação pré-renderizada para a interface do usuário da Web combinada com APIs e microsserviços é uma das plataformas mais rápidas e confiáveis disponíveis para aplicativos da Web avançados.

3 Os Desafios do Desenvolvimento Web Moderno

Em tecnologia, qualquer tipo de mudança – mesmo que seja gradual ou comece com projetos menores – requer grande esforço. Essas mudanças envolvem a avaliação de investimentos passados em infraestrutura, aquisição de novos conhecimentos e consideração cuidadosa de compensações de recursos.

Então, antes que possamos convencê-lo de que há valor em considerar o JAMstack como mais uma abordagem para desenvolvimento web, vamos gastar algum tempo investigando as abordagens mais difundidas para desenvolvimento web hoje e os desafios que elas apresentam. Simplificando, vejamos primeiro o risco de manter o status quo antes de mergulhar no JAMstack e nos benefícios de fazer uma troca.

3.1 Os inconvenientes de Arquiteturas Monolíticas

Os sites da Web têm sido tradicionalmente alimentados por arquiteturas monolíticas, com o frontend do aplicativo fortemente acoplado ao backend. Os aplicativos monolíticos são opinativos sobre como as tarefas no servidor ocorrem. As opiniões que eles impõem moldaram a forma como o desenvolvimento web é abordado e o ambiente no qual trabalhamos agora.

Populares e confiáveis como a maneira de fato de construir para a web, essas ferramentas tiveram um impacto profundo não apenas no desempenho dos sites, mas também na eficiência com que podem ser desenvolvidos e até na mentalidade dos desenvolvedores.

O aprisionamento criado por isso é tangível. Esses aplicativos ditam a abordagem de muitas facetas de um projeto de desenvolvimento da Web de maneiras que nem sempre são atraentes para a comunidade de desenvolvimento da Web em geral. A seleção de um aplicativo monolítico em detrimento de outro geralmente é baseada no tamanho de suas listas de recursos. A tendência de selecionar ferramentas com base no que há de melhor na classe, ou o mais popular no mercado, muitas vezes pode ignorar uma consideração fundamental: o que o projeto realmente precisa?

3.2 A complexidade que é necessária?

Ao nos fazermos essas perguntas regularmente, passamos a acreditar que existe um nível de super engenharia presente no desenvolvimento web hoje. Essa engenharia excessiva impede o uso de processos, tecnologias e arquiteturas comprovados e eficazes. Contudo, muitas vezes é possível adotar soluções mais adequadas

3.3 Flexibilidade Limitada

A liberdade de fazer um bom trabalho – projetar as experiências exatas que desejamos para nossos usuários – é regularmente comprometida por aplicativos monolíticos complexos.

Ao longo dos últimos anos, houve uma valorização crescente dos ganhos dramáticos de desempenho obtidos com a otimização do código front-end. No entanto, embora o reconhecimento do valor do forte desenvolvimento de front-end tenha crescido, as plataformas que os desenvolvedores usam geralmente não oferecem a liberdade de usar totalmente as habilidades de front-end.

As arquiteturas impostas por aplicativos monolíticos podem prejudicar nossa capacidade de utilizar as tecnologias mais recentes e até afetar coisas como nossos fluxos de trabalho de desenvolvimento. Isso inibe nossos esforços para proteger fundamentalmente aspectos importantes do desenvolvimento de software, como uma experiência de desenvolvimento saudável para liberar o potencial de nossas talentosas equipes de desenvolvimento.

3.4 Preocupações de desempenho

O desempenho do site é extremamente importante para o sucesso do conteúdo fornecido pela Internet. Existem muitos estudos que concluem que o desempenho e a conversão estão intimamente ligados. Um desses estudos concluiu que um atraso de um segundo no tempo de carregamento pode prejudicar a conversão de um site de comércio eletrônico em 7% (MAO, 2018; MARKOVIC, 2021).

Acontece que os aplicativos monolíticos raramente conduzem a um desempenho superior do site. Eles precisam gerar e entregar HTML toda vez que um novo visitante chega ao site. Isso diminui significativamente o tempo de carregamento da página (MAO, 2018; MARKOVIC, 2021).

Mas o desempenho não é ditado apenas pela velocidade. Como os aplicativos monolíticos são muito grandes, pode ser difícil definir os limites da arquitetura. O código é tão interconectado que corrigir um bug, atualizar uma biblioteca ou alterar uma estrutura em uma parte do aplicativo pode quebrar outra parte dele (MAO, 2018; MARKOVIC, 2021).

Armazenamento em cache é outra parte importante do desempenho geral do site — e é notoriamente difícil de acertar com um site dinâmico. Se um site for construído em um aplicativo monolítico, é possível que a mesma URL possa retornar conteúdo diferente dependendo de vários parâmetros, incluindo se um usuário está conectado ou se o site estava executando anteriormente um teste A/B. Ser capaz de oferecer uma experiência previsível aos usuários finais é vital (MAO, 2018; MARKOVIC, 2021).

3.5 Desafios

Como a arquitetura monolítica exige que a visualização da página seja gerada para cada visitante, a infraestrutura precisa ser dimensionada para antecipar o tráfego do site. Além de caro, é difícil acertar. As equipes acabam superprovisionando sua infraestrutura para evitar tempo de inatividade ou risco de falha porque não há uma separação clara entre a infraestrutura necessária para gerar e gerenciar o site e a necessária para servir o site (HOANG, 2020; PELTONEN et al., 2021).

Quando o mesmo aplicativo que cria as visualizações de página ou permite que os autores gerenciem o conteúdo também precisa ser dimensionado para lidar com picos de tráfego, não é possível desacoplar esses recursos para dimensionar e proteger cada parte da infraestrutura de acordo com suas necessidades (HOANG, 2020; PELTONEN et al., 2021).

Nesse cenário, as considerações que influenciam a arquitetura técnica de um site sofrem por serem agrupadas em um grande sistema. Na ciência da computação, reconhecemos que os custos de gravação e leitura podem ser muito diferentes e, no entanto, aplicativos monolíticos agrupam esses recursos no mesmo aplicativo, dificultando o design adequado (HOANG, 2020; PELTONEN et al., 2021).

A abordagem para projetar um sistema que permite centenas de milhões de operações de leitura é muito diferente daquela de um sistema que permite um número semelhante de operações de gravação. Então, é sensato combinar esses recursos na mesma infraestrutura e exigir que o perfil de risco de um recurso influencie o de outro? E esses recursos provavelmente encontrarão níveis de tráfego semelhantes? (HOANG, 2020; PELTONEN et al., 2021)

Existem maneiras de colocar distância entre os usuários que visitam o site e a complexidade que gera e entrega esse site. Sem tal separação, escala pode ser difícil de alcançar (HOANG, 2020; PELTONEN et al., 2021).

3.6 Problemas de segurança

Queremos garantir que, ao avaliar os motivos pelos quais os aplicativos monolíticos não atendem mais ao crescimento da Web, tenhamos o cuidado de

não insultar as centenas de milhares de equipes de desenvolvimento que optaram por implementá-los. Nós fomos esse time. Uma das maneiras de evitar isso é simplesmente olhar para os fatos. E quando falamos de segurança, os fatos são difíceis de descartar (MARKOVIC, 2021, BILLINGSLEY, 2020).

De acordo com estimativas recentes, o Wordpress alimenta 29% da web. Joomla e Drupal seguem como o segundo e terceiro sistemas de gerenciamento de conteúdo mais populares, respectivamente. que faz eles um alvo principal para o mal atores (MARKOVIC, 2021, BILLINGSLEY, 2020).

A disponibilidade desses produtos como serviços hospedados pode ajudar a nos isentar de algumas das responsabilidades de proteger a infraestrutura subjacente, mas em instâncias auto-hospedadas, devemos arcar com todo esse fardo. Em ambos os casos, existem muitos vetores de ataque, e a introdução de plugins de terceiros pode nos expor a mais riscos e tornar as coisas ainda mais difíceis de proteger (MARKOVIC, 2021, BILLINGSLEY, 2020).

Aplicativos monolíticos como Wordpress, Drupal e Joomla combinam cada componente e plug-in da arquitetura de um projeto da Web em uma única base de código. Por sua vez, cria uma enorme área de superfície para que o malware penetre. A área de superfície de ataque não é apenas extremamente grande, mas também é exposta toda vez que o site é criado, porque qualquer plug-in que o site usa deve ser executado sempre que a página for carregada para um novo visitante do site, aumentando o risco (MARKOVIC, 2021, BILLINGSLEY, 2020).

Com esse volume de distribuição, injetar malware em um único plug-in pode significar uma distribuição massiva. Um ataque recente fez com que 12 milhões de sites Drupal precisassem de patches de emergência (MARKOVIC, 2021, BILLINGSLEY, 2020).

E há mais uma pegadinha: em aplicativos monolíticos, os plug-ins são vinculados diretamente à estrutura principal. Por serem notoriamente inseguros e exigirem atualizações frequentes (e muitas vezes com erros), os plug-ins atualizados às pressas correm o risco de quebrar todo o site. Os mantenedores podem escolher entre os patches de segurança e o risco de quebrar a funcionalidade do site (MARKOVIC, 2021, BILLINGSLEY, 2020).

3.7 O risco de continuar com o uso da arquitetura web tradicional

Desenvolvedores e tomadores de decisão de negócios reconheceram os riscos de continuar desenvolvendo projetos da Web usando aplicativos monolíticos. Mas o risco de mudança muitas vezes supera o risco de permanecer o mesmo. Para grandes empresas, o investimento maciço em arquitetura e talento combina com o medo de abrir mão da flexibilidade, recursos e ferramentas de um ecossistema mais maduro. Para pequenas empresas, a reconstrução da infraestrutura da Web retira recursos das iniciativas críticas de crescimento (MARKOVIC, 2021, BILLINGSLEY, 2020).

Os primeiros sussurros de mudança vieram como uma onda de blogs pessoais, sites de documentação de projetos de código aberto e projetos com prazo determinado, como eventos e sites de conferência, migrados para o JAMstack (MARKOVIC, 2021, BILLINGSLEY, 2020).

Esses sussurros ficaram mais altos à medida que sites maiores com maior número de audiência e mais conteúdo também começaram a adotar o JAMstack, desafiando limitações anteriormente aceitas ao fornecer comentários, conteúdo de assinatura, comércio eletrônico e muito mais (MARKOVIC, 2021, BILLINGSLEY, 2020).

À medida que a população mundial está mais e mais online, é essencial que forneçamos sites que possam lidar com grandes picos de tráfego, funcionar de forma rápida e previsível em quaisquer condições de tráfego, oferecer suporte a diferentes idiomas e localizações e ser acessível e utilizável em todos os dispositivos que conhecemos e muitos que nós não. Além disso, os sites precisam fazer tudo isso enquanto estão seguros e protegidos contra ataques maliciosos inerentes a uma rede global aberta (MARKOVIC, 2021, BILLINGSLEY, 2020).

4 Planejamento para o JAMstack

Este capítulo abrange a configuração de ponta a ponta do JAMstack e destaca as escolhas que se fará ao longo do caminho para criar a configuração

apropriada para sua equipe. Esses decisões incluem a seguinte (MARKOVIC, 2021; OROSZ, 2020):

- Como se gerencia o projeto/código
- Onde se armazena o seu conteúdo
- Qual software irá construir seu site
- Como o processo de construção é automatizado
- Onde o site será publicado
- Quais serviços e APIs o site ao vivo usará

4.1 Configuração do Projeto

Sem servidor ou banco de dados para configurar, um projeto JAMstack é basicamente uma estrutura de pastas simples cheia de arquivos de texto. O conteúdo do projeto JAMstack geralmente inclui o seguinte (MARKOVIC, 2021; OROSZ, 2020):

- Uma pasta de origem para armazenar seus arquivos de conteúdo
- Uma pasta para layouts e modelos
- Um arquivo de configuração contendo as configurações para o processo de compilação
- Um gerador de site estático, geralmente adicionado como uma dependência
- Uma pasta de destino para salvar a saída final

Podemos gerenciar todo projeto localmente, diretamente em seu computador, mas é sempre melhor começar configurando um Git hospedado repositório. Mesmo se o autor do site iniciar um repositório lhe dará um backup online, armazenará o histórico do projeto e, eventualmente, permitirá que outros contribuam. E em um momento, o autor também verá como um repositório hospedado é a chave para acionar novas compilações sempre que uma mudança é enviada (MARKOVIC, 2021; OROSZ, 2020).

Raramente se configuraria sua pasta de projeto inteira do zero porque a maioria dos geradores de sites tem projetos de amostra que se pode baixar do

GitHub como ponto de partida. Mas antes de falarmos sobre geradores de sites, vamos abordar as várias maneiras de gerenciar conteúdo no JAMstack(MARKOVIC, 2021; OROSZ, 2020).

4.2 Estratégias para gerenciar Contente

Todo site começa com conteúdo. Em sistemas de sistema de gerenciamento de conteúdo (CMS), como Drupal ou Wordpress, os sites gerenciam o conteúdo em um banco de dados e os editores usam uma interface de administração para gravar e salvar o conteúdo no banco de dados (MARKOVIC, 2021; OROSZ, 2020).

4.3 Arquivos de texto

Para muitos sites JAMstack, o conteúdo não poderia ser mais direto: é simplesmente uma pasta cheia de arquivos que reside diretamente no projeto com os modelos, plug-ins e outros ativos. A criação de um novo arquivo no diretório de conteúdo criará uma nova página após a publicação do site, e as alterações são feitas editando os arquivos de conteúdo em um editor de texto e, em seguida, confirmando as alterações no repositório Git. O nome de cada arquivo e a pasta em que ele se encontra determinarão seu caminho de URL após a geração do site (UTOMO, 2020; HOANG, 2020).

Como é um desses arquivos de conteúdo? Pode ser HTML direto, embora geralmente emparelhado apenas com as partes exclusivas da página. Elementos comuns, como cabeçalhos e rodapés, são adicionados pelo gerador do site durante a etapa de construção (UTOMO, 2020; HOANG, 2020).

No entanto, criar conteúdo de site em HTML puro ainda pode ser tedioso. É por isso que o Markdown uma sintaxe muito simplificada, tornou-se a opção mais popular para escrever conteúdo. Se usamos o Slack e colocou uma palavra com asteriscos em *negrito*, usamos uma forma modificada de markdown (UTOMO, 2020; HOANG, 2020).

Os CMSs mais comuns têm configurações adicionais para cada página; por exemplo, qual layout usar ou qual categoria e tags a página deve ser atribuída.

Para armazenar detalhes como esses, os geradores de sites estáticos padronizaram uma seção superior de cada arquivo chamada metadados. Muitas vezes escrito em YAML, é uma simples coleção de chave/valor de configurações e outros dados que se aplicam à página (UTOMO, 2020; HOANG, 2020).

É provável que nem todos os colaboradores do seu site sejam desenvolvedores, e usar Git e arquivos de texto pode ser uma experiência desconhecida. Esta é realmente a razão pela qual CMSs baseados em banco de dados como o Wordpress foram originalmente criados. No entanto, provavelmente também experimentamos como um CMS complexo pode criar uma separação artificial entre os editores de conteúdo que trabalham no banco de dados e os desenvolvedores que criam o código e os modelos (UTOMO, 2020; HOANG, 2020).

Como cada página da web acaba sendo uma combinação de conteúdo e marcação, seria bom se pudessemos gerenciar ambos pelo mesmo processo

Existe agora uma nova geração de software CMS que faz exatamente isso, gerenciando conteúdo como arquivos no Git em vez de um banco de dados. Normalmente incluído em seu projeto como um aplicativo de página única (apenas um arquivo HTML com algumas dependências de JavaScript), um CMS baseado em Git permite que os usuários efetuem login em uma interface de administração familiar para editar conteúdo e visualizar e salvar alterações (UTOMO, 2020; HOANG, 2020).

Aqui está a parte inteligente: uma alteração salva no CMS é realmente feita como uma contribuição do Git nos bastidores. Isso é poderoso porque os desenvolvedores e editores estão trabalhando no mesmo fluxo de trabalho baseado em Git, modificando os mesmos arquivos usando o método que for mais confortável para eles (UTOMO, 2020; HOANG, 2020).

4.4 CMS *Headless*

Esta é uma categoria de serviços hospedados que fornecem uma rica interface online para gerenciamento e edição de conteúdo (UTOMO, 2020; HOANG, 2020).

“Headless” refere-se à ideia de que esses serviços gerenciam apenas o conteúdo, deixando para o desenvolvedor determinar como e onde o site é construído e hospedado. Projetos maiores usam um CMS headless como um armazenamento central de conteúdo por trás de toda a coleção de sites e aplicativos móveis (UTOMO, 2020; HOANG, 2020).

Todos os serviços CMS headless fornecem uma API que é usada para extrair o conteúdo mais recente durante a etapa de compilação (e também usando JavaScript dinamicamente à medida que o site é executado) (UTOMO, 2020; HOANG, 2020).

Por exemplo, usamos um CMS headless para sua equipe de recrutamento gerenciar anúncios de emprego. O gerador de site estático consultaria a API durante a etapa de construção, recebendo uma lista dos anúncios de emprego como JSON. Em seguida, usaria modelos para criar uma página HTML exclusiva para cada postagem, além, é claro, de uma página de índice listando todas as postagens juntas (UTOMO, 2020; HOANG, 2020).

4.5 CMS auto-hospedado

A execução do JAMstack não impede que usemos algo mais tradicional como Wordpress ou Drupal para gerenciar conteúdo. Em versões recentes, esses dois CMSs populares adicionaram APIs para acessar o conteúdo que eles contêm. Assim, assim como o serviço CMS headless mencionado acima, podemos conectar seu conteúdo ao seu site JAM-stack usando a API (UTOMO, 2020; HOANG, 2020).

Dessa forma, é possível continuar gerenciando o conteúdo em um CMS familiar, mas agora pré-cria o site para acelerar drasticamente o desempenho, servindo páginas renderizadas de um CDN. E Como o Wordpress e o Drupal

não precisariam mais ser voltados para o público, temos muito mais opções para mantê-los seguros (UTOMO, 2020; HOANG, 2020).

4.6 Escolhendo um gerador de sites

Com o conteúdo gerenciado usando uma ou mais das técnicas descritas, é hora de pensar em como ele é transformado em HTML de produção.

No início, os geradores de sites eram quase sinônimo de Jekyll, o primeiro gerador de sites a ganhar maior tração e popularidade. Jekyll foi criado por Tom Preston-Werner, cofundador do GitHub, usando a linguagem de programação Ruby (ATTARDI, 2020).

Hoje, os geradores de sites estão disponíveis em quase todas as linguagens, incluindo Ruby, Python, PHP, JavaScript e Go. O site staticgen.com é um diretório com curadoria da comunidade de geradores de sites estáticos, com mais de 30 idiomas representados. A maioria dos geradores de sites são gratuitos e de código aberto (ATTARDI, 2020).

Talvez a parte mais difícil de começar com o JAMstack seja escolher o gerador de site apropriado na lista de opções em constante expansão. O site staticgen.com mostra a popularidade de cada gerador, mas além do que os outros pensam, existem alguns fatores reais que devemos considerar para dar sabor à sua experiência com o JAMstack (ATTARDI, 2020).

4.7 Um : O Tipo do site

Veremos que cada gerador é criado por seus autores com um tipo específico de site ou aplicativo em mente. Alguns geradores são projetados para sites com taxonomia complexa e milhares de páginas, outros especificamente para blogs, outros para documentação, outros para aplicativos da web progressivos (PWAs) e aplicativos altamente interativos para os quais quase todo o conteúdo e navegação são carregados e manipulados via JavaScript. Nosso conselho é verificar projetos de amostra criados com cada

gerador de site estático para ver se eles correspondem aos seus próprios resultados pretendidos (ZAMMETTI, 2020; HOANG, 2020).

4.8 Dois : A Programação Linguagem Usado

É muito possível usar um gerador de sites sem conhecer a linguagem de programação usada para criá-lo. Por exemplo, mesmo que não saibamos nada sobre a linguagem de programação Go, Hugo é uma ótima escolha porque podemos baixá-lo e instalá-lo como um aplicativo binário sem mais nada para configurar e nenhum conhecimento de programação necessário (ZAMMETTI, 2020; HOANG, 2020).

Jekyll também não requer nenhuma experiência específica em Ruby para fazer bom uso dele, mas primeiro precisamos ter uma instalação de Ruby funcionando para executá-lo em seu computador. Gatsby, outro gerador de sites popular que usa o framework React JavaScript, requer que tenhamos uma versão recente do Node.js instalada. Entendemos: um pouco de experiência com as ferramentas em torno da linguagem pode ajudar (ZAMMETTI, 2020; HOANG, 2020).

Não é uma má ideia considerar um gerador de site escrito em um idioma que sua equipe considere familiar. À medida que seu site se torna mais complexo, Podemos encontrar a necessidade de seus próprios plug-ins e transformações personalizados. A familiaridade com a linguagem do gerador pode ajudá-lo a modificar e estender sua funcionalidade (ZAMMETTI, 2020; HOANG, 2020).

4.9 Três : A modelagem Sintaxe

A maior parte do desenvolvimento do JAMstack será gasto criando e modificando modelos HTML, CSS e JavaScript. Dê uma olhada nas linguagens de template que seu gerador de site estático escolhido suporta. Todas as linguagens de modelagem aumentam a marcação HTML tradicional com novas funcionalidades, como blocos de código reutilizáveis, instruções if e loops. No entanto, a sintaxe exata usada varia e é em grande parte uma questão de

preferência pessoal. Alguns desenvolvedores preferem o estilo minimalista de uma linguagem como Jade ou Pug - ambos dispensam colchetes e tags de fechamento em favor do recuo. Outros desenvolvedores preferem linguagens de modelo como Handlebars e Liquid, que parecem o mais próximo possível do HTML tradicional (ZAMMETTI, 2020; HOANG, 2020).

4.10 Quatro: Velocidade

A velocidade com que seu gerador de sites pode construir cada página se torna importante à medida que seu conteúdo cresce. Um site com um punhado de páginas pode levar apenas milissegundos para ser construído, enquanto um site com arquivos de conteúdo pode levar vários minutos para ser concluído. Como cada alteração publicada executará o processo de compilação, a velocidade é uma consideração importante (MARKOVIC, 2021; ZAMMETTI, 2020; HOANG, 2020).

Hugo é notoriamente rápido na construção de páginas de sites, graças à arquitetura multi-thread do Go. Outros geradores de sites, por natureza de seus objetivos de design, criam HTML estático e um pacote JavaScript para cada página. O JavaScript será usado para reduzir os tempos de carregamento de um usuário navegando pelo site depois que ele estiver em uma CDN, mas o trabalho extra envolvido aumenta significativamente o tempo de compilação (ZAMMETTI, 2020; HOANG, 2020).

4.11 Cinco: Desenvolvedor Ferramentas

O objetivo principal de um gerador de site é transformar conteúdo em HTML, mas como sabemos, um site sempre contém outros ativos como CSS, imagens e JavaScript. O desenvolvimento web frontend moderno está começando a se parecer muito com o desenvolvimento de software tradicional, com cadeias de ferramentas para preparar e compilar ativos para produção (MARKOVIC, 2021; ZAMMETTI, 2020; HOANG, 2020).

Antes de publicar o site, as imagens geralmente precisam ser compactadas, JavaScript transpilado e reduzido e CSS combinado ou convertido de formatos como SASS e LESS. Recentemente, também se tornou comum inserir CSS e imagens menores diretamente no HTML para o benefício de desempenho do navegador, não precisando fazer uma solicitação extra para buscar esses itens (MARKOVIC, 2021; ZAMMETTI, 2020; HOANG, 2020).

Portanto, a construção de um site JAMstack geralmente é feita dessas duas etapas: criar o HTML e compilar esses ativos de produção. Alguns geradores de sites se preocupam apenas com o HTML, deixando a produção do restante dos ativos para outro utilitário como Gulp ou Webpack. Outros geradores parecem ser mais holísticos e, na verdade, incluem recursos para lidar com a compilação de CSS JavaScript e imagens – especialmente geradores de sites construídos em estruturas JavaScript como Nuxt (Vuejs) ou Gatsby (React). Encontramos muitos geradores que realmente usam ferramentas populares como o Webpack sob o capô (MARKOVIC, 2021; ZAMMETTI, 2020; HOANG, 2020).

Linting também é comum. Este é o processo de execução automática de testes em HTML, JavaScript e CSS para verificar se há erros. Essa é uma maneira fantástica de descobrir problemas antes que eles o mordam na produção. Os linters também costumam ser específicos sobre a formatação, garantindo que todos que contribuem sigam os mesmos padrões (MARKOVIC, 2021; ZAMMETTI, 2020; HOANG, 2020).

Por fim, os desenvolvedores que trabalham localmente costumam visualizar as alterações localmente em uma sessão rápida de edição de arquivos e visualização de alterações em um navegador. Para dar suporte a isso, a maioria dos geradores de sites inclui um pequeno servidor da Web que pode permitir que veiculemos o site localmente para teste. Melhor ainda é o recarregamento a quente, que é a capacidade do navegador ser atualizado imediatamente assim que uma alteração é salva, sem a necessidade de clicar em atualizar ou perder o estado atual do seu aplicativo (MARKOVIC, 2021; ZAMMETTI, 2020; HOANG, 2020).

Recomendar um gerador de sites é um pouco como recomendar um dispositivo de computação: realmente depende do que o autor prefere, com o que está familiarizado e como pretende usá-lo. Não há realmente nenhuma

maneira clara de ser prescritivo, infelizmente. Além disso, há um trabalho muito ativo sendo feito nesse espaço – no momento em que qualquer lista “Best Of” é publicada, ela já está desatualizada (MARKOVIC, 2021; ZAMMETTI, 2020; HOANG, 2020).

Felizmente, tentar uma primeira execução de qualquer gerador de sites geralmente leva apenas alguns minutos. Nós realmente aconselhamos experimentar vários e sempre navegar na documentação para quaisquer objetivos de design, porque eles informarão sobre os problemas que os criadores estão trabalhando para resolver (MARKOVIC, 2021; ZAMMETTI, 2020; HOANG, 2020).

4.12 Adicionando automação

Dedicar tempo para adicionar automação de compilação é o que torna o desenvolvimento do JAMstack tão produtivo e agradável (UTOMO, 2020).

Todos os geradores de sites têm um comando de compilação que pode ser executado a partir do terminal. Os ativos finais geralmente são compilados em uma pasta chamada dist ou build e podem ser carregados em praticamente qualquer servidor ou CDN. Certamente é possível construir e carregar ativos manualmente — especialmente no início. No entanto, à medida que desenvolvemos mais e envolvemos mais membros da equipe, descobriremos que essa abordagem não é dimensionada (MARKOVIC, 2021; UTOMO, 2020).

Mencionamos anteriormente que podemos usar um repositório Git hospedado para seu projeto para fornecer um backup online e facilitar a colaboração. Mas agora exploramos o terceiro motivo: acionar compilações automatizadas por meio de webhooks (MARKOVIC, 2021;UTOMO, 2020).

GitHub, GitLab e Bitbucket são os três serviços de hospedagem de repositório mais populares e todos eles suportam webhooks. Webhooks são uma maneira simples de qualquer evento em uma plataforma (como um push de código) acionar um HTTP POST para uma URL externa. Ao configurar seu webhook, podemos determinar o URL usado. A carga útil do POST conterá informações JSON sobre o evento. O servidor que recebe o webhook usa essas informações para determinar qual ação tomar (MARKOVIC, 2021;UTOMO, 2020).

Para sites JAMstack, a ação mais comum é acionar uma nova compilação do site quando algo é adicionado ou modificado por meio de um push para a origem do Git. Alguns serviços de implantação mais avançados podem até criar visualizações de pull requests em URLs dedicados. Dessa forma, as equipes podem navegar e testar as alterações antes de mesclá-las. Também é possível que cada branch de um repositório seja construído separadamente em URLs únicos. Esta é uma abordagem poderosa para ambientes de teste e trabalhar em ramificações de recursos (MARKOVIC, 2021;UTOMO, 2020).

Idealmente, seu processo de compilação automatizado fará o seguinte (MARKOVIC, 2021;UTOMO, 2020):

1. Ouça as notificações de alterações no seu repositório Git (geralmente via webhooks).
2. Prepare o ambiente de compilação e busque quaisquer dependências necessárias.
3. Busque dados remotos de APIs (conforme necessário).
4. Construir o site e preparar ativos
5. Publique o site final em um CDN.

4.13 Usando GitHub Pages e GitLab Pages

Caso o repositório esteja hospedado no GitHub ou GitLab, considere usar GitHub Pages ou GitLab Pages. Ambas são opções leves que se conectam aos seus repositórios e permitem que criemos e hospedemos automaticamente um site estático. Ambos suportam personalizadas domínios (ZAMMETTI, 2020).

GitHub Pages foi lançado originalmente para facilitar o desenvolvimento de sites e documentação para os projetos de código aberto que eles hospedam. Se precisamos de um site simples para explicar seu projeto, é uma ótima opção. Observe que o GitHub Pages é compatível apenas com o gerador de sites Jekyll e apenas com plug-ins específicos (ZAMMETTI, 2020).

GitLab Pages oferece suporte a uma variedade maior de geradores de sites estáticos e podemos até criar seu próprio pipeline de integração contínua

(CI) para implantar e hospedar seu site. O GitLab também pode ser auto-hospedado se for um requisito ou interesse da sua organização (ZAMMETTI, 2020).

Se tivermos infraestrutura existente, poderá decidir executar seu próprio processo de compilação hospedado em um servidor, máquina virtual ou contêiner. Existem muitas ferramentas de integração contínua/desenvolvimento contínuo (CI/CD) que podemos aplicar para automatizar a pilha JAM. Sejá estivermos executando a infraestrutura de CI/CD, poderemos conectar a publicação do JAMstack aos seus fluxos de trabalho existentes (ZAMMETTI, 2020).

4.14 Usando um serviço de implantação hospedado

Como configurar um sistema de compilação robusto e resiliente pode ser difícil, uma nova categoria de serviço surgiu: serviços de compilação hospedados/integração contínua. Esses serviços geralmente têm aplicativos disponíveis para GitHub e GitLab, tornando a integração com seu repositório bastante simples. Em alguns, podemos selecionar seu próprio destino de implantação (como Amazon Web Services Simple Storage Service); em outros, os serviços CDN e Domain Name System (DNS) são integrados, fornecendo tudo o que precisamos para construir e hospedar o respectivo aplicativo (UTOMO, 2020; ZAMMETTI, 2020).

A maioria dos serviços de compilação geralmente cria um contêiner temporário, configura seu ambiente, obtém todas as dependências necessárias, executa a compilação, implanta os resultados e descarta o contêiner. É importante que o serviço de compilação esteja ciente e ofereça suporte ao seu gerador de sites, ambiente de idioma e outras ferramentas envolvidas na criação de seu site. Vamos querer manter arquivos de configuração de dependência como Gemfile, package.json e composer.php precisos e atualizados (ZAMMETTI, 2020).

4.15 Selecionando um CDN

Mesmo com todos os avanços modernos em tecnologias de rede, o desempenho de sites e aplicativos ainda sucumbe aos dois inimigos mais antigos da internet: tempo e espaço. Felizmente, a abordagem JAMstack de publicar diretamente em CDNs ajuda a resolver ambos. Primeiro, o tempo necessário para começar a veicular o conteúdo é muito melhor com a pré-renderização toda a marcação, como discutimos anteriormente. E segundo, a quantidade de distância física (e saltos de rede) que o conteúdo deve percorrer no caminho para os usuários também é reduzida, graças à natureza geograficamente distribuída das CDNs (ZAMMETTI, 2020).

Uma CDN é uma coleção global de nós de borda localizados em diferentes data centers, todos interconectados com largura de banda rápida e de alta qualidade. Quando um usuário carrega conteúdo de uma CDN, ele sempre está conectado ao nó de borda mais próximo possível. E, é claro, distribuir seu site em vários nós de borda também ajuda na disponibilidade. Se um ou mais nós de borda, interconexões de rede ou até mesmo datacenters falharem, seu site permanecerá disponível (UTOMO, 2020; ZAMMETTI, 2020).

Aqui estão três considerações para escolher uma CDN para seu aplicativo da web (UTOMO, 2020; ZAMMETTI, 2020):

- Pontos de presença: publicar em uma CDN geralmente significa que o conteúdo do seu site fica disponível para um mínimo de 10 ou 15 locais em todo o mundo. Desejaremos uma CDN com pontos de presença próximos ao seu público, portanto, se tivermos usuários em Tóquio, eles experimentarão seu site veiculado em Tóquio.
- Atualizações instantâneas: os CDNs costumavam funcionar como camadas de cache bastante não inteligentes na frente dos servidores da web. Embora eles certamente acelerassem o desempenho do aplicativo, documentos e imagens na CDN sempre recebiam configurações bastante longas de tempo de vida (TTL). Quando o cache foi definido para durar horas ou mais, as atualizações foram atrasadas e limpar o CDN para abrir caminho para novos conteúdos se mostrou bastante difícil. Muitas vezes, era um equilíbrio delicado entre permitir conteúdo mais dinâmico com atualizações frequentes versus usar um cache mais

longo para reduzir a carga nos servidores de origem. As CDNs modernas agora permitem invalidação instantânea, substituindo arquivos em milissegundos e borrando as linhas entre conteúdo estático e atualizações dinâmicas. Se uma atualização quase instantânea do cache da CDN puder ser garantida toda vez que o site for implantado, os TTLs e o conteúdo obsoleto não serão mais uma preocupação. Hoje, em vez de extrair conteúdo periodicamente de um servidor de origem, as atualizações podem ser enviadas para CDNs toda vez que o site muda. Esse avanço nos permite publicar diretamente em CDNs e ignorar servidores – um componente crítico do JAMstack.

- Vários provedores: alguns CDNs localizam seus servidores de borda com vários provedores de nuvem, aumentando a durabilidade do seu aplicativo. Mesmo que uma interrupção completa do datacenter seja rara, elas acontecem. Também não é incomum que um provedor sofra um ataque de negação de serviço distribuído (DDoS) em relação a uma instalação ou rede específica. Certifique-se de que seu provedor de CDN não seja apenas distribuído em todo o mundo, mas também entre provedores. A hospedagem em um provedor desse tipo oferece o poder de uma “nuvem de nuvens” para a configuração mais resiliente possível.
- Fornecendo Funcionalidade: Mesmo que um site JAMstack possa ser nada mais do que uma coleção de ativos estáticos, conectar seu aplicativo ao vasto mundo de serviços baseados em API é o que mostra o verdadeiro poder e promessa do JAMstack como uma arquitetura para o mundo real formulários.

Não faz muito tempo que autenticação, processamento de pagamentos e comércio eram todas as preocupações que se precisava para hospedar e gerenciar a si mesmo. Mas hoje, cada um desses (e muitos outros recursos) agora são consumíveis como APIs (UTOMO, 2020; ZAMMETTI, 2020).

Mas quão caro será? E se o serviço cair? Quão difícil seria arrancar e substituir? Essas perguntas são comuns ao avaliar qualquer serviço hospedado

e pode parecer assustador confiar em serviços críticos a terceiros. Mas considere que todas as perguntas acima também surgirão inevitavelmente para equipes que constroem ou gerenciam tecnologias internamente (UTOMO, 2020; ZAMMETTI, 2020).

Tudo se resume a escolha e flexibilidade, e a mudança para arquiteturas desacopladas e orientadas por API (e longe de aplicativos monolíticos) oferece às equipes a flexibilidade de decidir o que consumir e o que construir. Essa é uma construção poderosa porque cada camada da pilha - do conteúdo ao comércio e ao front-end - agora pode ser examinada e escolhida de forma independente (UTOMO, 2020; ZAMMETTI, 2020).

4.16 Vamos ver isso em ação.

Considere as necessidades de uma loja de comércio eletrônico. Primeiro, o site precisa ser rápido e confiável, mesmo sob carga pesada. O espírito do JAMstack de páginas estáticas publicadas em uma CDN global garante notas incrivelmente altas em velocidade e confiabilidade. Sem gerenciar uma frota de servidores, é possível acompanhar o desempenho oferecido pelos gigantes do comércio eletrônico (UTOMO, 2020; ZAMMETTI, 2020).

Mas igualmente importante, obtém-se o controle total sobre todo o frontend, até o último e pixel. O desenvolvedor é livre para projetar e desenvolver qualquer página que possa imaginar, usando a linguagem, ferramentas e modelos de sua escolha. Sua equipe pode criar experiências personalizadas para suas especificações exatas. Essa é uma abordagem muito diferente de combater a saída de marcação por um aplicativo de comércio eletrônico hospedado em servidor tradicional (UTOMO, 2020; ZAMMETTI, 2020).

O comércio, porém, não é estático. É necessário de uma maneira de gerenciar o conteúdo (neste caso, itens à venda), pesquisa avançada, gerenciar o inventário disponível e permitir que os clientes façam compras. No JAMstack, usamos APIs para integrar serviços sob medida para cada componente (UTOMO, 2020; ZAMMETTI, 2020).

Adicionar um novo item no Contentful aciona uma nova compilação do site usando a automação de compilação do Netlify. Durante a etapa de compilação, o Netlify busca os dados de armazenamento mais recentes do Contentful chamando a API. Esses dados são alimentados em um gerador de sites (como Gatsby) para renderizar cada uma das páginas (UTOMO, 2020; ZAMMETTI, 2020).

Também durante a etapa de construção, as APIs são usadas para enviar dados para Algolia, o provedor de pesquisa, para que o índice de pesquisa possa ser atualizado à medida que o catálogo de produtos muda (UTOMO, 2020; ZAMMETTI, 2020).

À medida que os visitantes navegam no site, as páginas pré-renderizadas são extraídas diretamente de um CDN, sem nenhum processo de servidor em execução em nenhum lugar para ser encontrado. Então, como o site é feito interativo? Aproveitando o poder do JavaScript para chamar APIs diretamente do navegador (UTOMO, 2020; ZAMMETTI, 2020).

Quando um comprador digita um termo de pesquisa, ele é enviado para Algolia via Ajax, e Algolia retorna JSON com os resultados. Javascript é usado para exibir esses resultados da maneira que quisermos integrá-los ao nosso aplicativo (UTOMO, 2020; ZAMMETTI, 2020).

Depois que o comprador clica no botão comprar, a intenção de compra é enviada para um provedor chamado CommerceLayer, cujo trabalho é alimentar o carrinho de compras e a experiência de checkout. Novamente, isso acontece usando JavaScript para chamar APIs. Assim como o Contentful, o CommerceLayer também é headless e não faz exigências sobre a aparência e o comportamento do aplicativo (UTOMO, 2020; ZAMMETTI, 2020).

Aí está: toda uma experiência de comércio construída pela interconexão de componentes modulares por meio de chamadas de API. Como não há infraestrutura de back-end para configurar ou gerenciar, todo o nosso tempo pode ser gasto criando o aplicativo de front-end. E como tudo o que estamos usando é um serviço hospedado, um primeiro protótipo pode ser criado em cerca de um ou dois dias (UTOMO, 2020; ZAMMETTI, 2020).

É por isso que devemos ter ouvido falar que o JAMstack fornece “superpoderes” para equipes de front-end. Hoje, podemos conectar funcionalidades avançadas de nível empresarial usando serviços de API como

os aqui. Mas, novamente, nada é prescritivo no JAMstack e houve uma explosão de novas ofertas de serviços de API projetadas para potencializar praticamente qualquer aspecto do seu projeto. Na verdade, encontraremos muitos players tradicionais como Shopify agora oferecem APIs para que eles também possam ser consumidos sem cabeça. A explosão de APIs fala com o impulso do JAMstack como a arquitetura da web interconectada (UTOMO, 2020; ZAMMETTI, 2020).

4.17 Simplificando Sistemas e Simplificando o Pensamento

Graças aos inovadores que construíram camadas de abstração sobre os bits e bytes de baixo nível que os computadores usam para operar, a quantidade de trabalho que é possível por um único desenvolvedor agora é astronômica (UTOMO, 2020; ZAMMETTI, 2020).

As abstrações, no entanto, são vazadas. Às vezes, o nível inferior da pilha de repente se torna conhecido. Podemos estar trabalhando em um alto nível de programação orientada a objetos, desenvolvendo um aplicativo orientado a banco de dados com um Mapeamento Relacional de Objeto (ORM) quando de repente um problema de gerenciamento de memória de baixo nível, profundo em uma dependência compilada, abre um estouro de buffer que um usuário mal-intencionado inteligente pode abusar para sequestrar o banco de dados e assumir completamente o controle (UTOMO, 2020; ZAMMETTI, 2020).

A menos que encontremos maneiras de construir firewalls sólidos entre as diferentes camadas da pilha, esses tipos de problemas sempre serão capazes de atravessar as camadas inferiores ou os pontos distantes da cadeia de dependência (UTOMO, 2020; ZAMMETTI, 2020).

Felizmente, o JAMstack cria vários desses firewalls para gerenciar a complexidade e focar a atenção em partes menores isoladamente (UTOMO, 2020; ZAMMETTI, 2020).

4.18 Melhor compreensão e agilidade mental

Com a renderização do lado do servidor, todas as camadas possíveis da pilha estão sempre envolvidas em qualquer uma das solicitações que um usuário faz a um site ou aplicativo. Para construir software seguro e de alto desempenho, os desenvolvedores devem entender profundamente todas as camadas pelas quais as solicitações fluem, incluindo plug-ins de terceiros, drivers de banco de dados e detalhes de implementação de linguagens de programação (MARKOVIC, 2021).

Desacoplar completamente o estágio de construção do tempo de execução do sistema e publicar ativos pré-fabricados constrói uma parede inquebrável entre o ambiente de tempo de execução com o qual os usuários finais interagem e o espaço onde o código é executado. Isso torna muito mais fácil projetar, desenvolver e manter o tempo de execução isolado do estágio de construção (MARKOVIC, 2021).

Separar APIs em microsserviços menores com um propósito bem definido significa que nossa capacidade de compreender as particularidades de cada serviço isoladamente se torna muito mais simples. As fronteiras entre os serviços são completamente claras (MARKOVIC, 2021).

Isso nos ajuda a estabelecer modelos mentais mais claros do sistema como um todo, ao mesmo tempo em que nos libertamos da necessidade de entender as complexidades internas de cada serviço e sistema individual. Como resultado, a carga de desenvolvimento e manutenção do sistema pode ser significativamente aliviada. As áreas para as quais devemos direcionar nossa atenção podem ser mais focadas com a confiança de que os limites entre os serviços utilizados são robustos e bem definidos (MARKOVIC, 2021).

Não há necessidade de entender as características de tempo de execução do JavaScript em diferentes navegadores, bem como os fluxos de uma API ao mesmo tempo (MARKOVIC, 2021).

4.19 O desenvolvedor não precisa ser especialista em tudo

Nos últimos anos, a complexidade das arquiteturas front-end explodiu à medida que as APIs dos navegadores evoluíram e os padrões HTML e CSS cresceram. No entanto, é muito difícil para a mesma pessoa ser especialista em desempenho JavaScript do lado do cliente e desenvolvimento do lado do

servidor, restrições de consulta de banco de dados, otimização de cache e operações de infraestrutura (MARKOVIC, 2021).

Quando separamos o back-end e o front-end, é possível que os desenvolvedores se concentrem em apenas uma área. Podemos executar seu frontend localmente com um servidor de desenvolvimento com atualização automática falando diretamente com uma API de produção e tornar a alternância entre APIs de teste, produção ou desenvolvimento tão simples quanto definir uma variável de ambiente (MARKOVIC, 2021).

E com microsserviços muito pequenos e super focados, a camada de serviço se torna muito mais reutilizável e geralmente aplicável do que quando temos um grande aplicativo monolítico cobrindo todas as nossas necessidades (MARKOVIC, 2021).

Isso significa que estamos vendo um ecossistema muito mais amplo de APIs prontas para autenticação, comentários, comércio eletrônico, pesquisa, redimensionamento de imagens e assim por diante. Podemos usar ferramentas de terceiros para terceirizar a complexidade e ficar tranquilos sabendo que esses provedores têm equipes altamente especializadas focando exclusivamente em seu espaço de problemas (MARKOVIC, 2021).

4.20 Reduzir Mudança Peças no Tempo de execução

Quanto mais pré-compilarmos, mais poderemos implantar como marcação pré-confeccionada sem a necessidade de execução de código dinâmico em nossos servidores durante um ciclo de solicitação, e melhor seremos. Executar código zero sempre será mais rápido do que executar algum código. O código zero sempre será mais seguro do que a menor quantidade de código. Os ativos que podem ser atendidos sem partes móveis sempre serão mais fáceis de dimensionar do que até mesmo o programa dinâmico mais otimizado (MARKOVIC, 2021;UTOMO, 2020).

Menos peças móveis em tempo de execução significam menos coisas que podem falhar em um momento crítico. E quanto mais distância pudermos colocar entre nossos usuários e a complexidade inerente aos nossos sistemas, maior será nossa confiança de que eles experimentarão o que pretendíamos. Enfrentar essa complexidade em tempo de construção, em um ambiente que

não afetará os usuários, nos permite expor e resolver quaisquer problemas que possam surgir com segurança e sem impacto negativo (MARKOVIC, 2021;UTOMO, 2020).

A única razão para executar o código do lado do servidor no momento da solicitação deve ser que absolutamente não podemos evitá-lo. Quanto mais nos libertarmos disso, melhor será a experiência para nossos usuários, equipes de operações e nossa própria capacidade de entender os projetos em que estamos trabalhando.

4.21 Custos

Há uma variedade de custos associados ao design, desenvolvimento e operação de sites e aplicativos, e eles são, sem dúvida, influenciados pela pilha que escolhemos. Embora os custos financeiros sejam geralmente mais óbvios, também devemos considerar os custos para a experiência, criatividade e inovação do desenvolvedor (MARKOVIC, 2021;UTOMO, 2020).

4.22 Custos financeiros

Qualquer projeto de desenvolvimento web de escala significativa provavelmente incluirá um exercício para estimar os níveis de tráfego previstos ou de destino e, em seguida, um plano para a infraestrutura de hospedagem necessária para atender a esse tráfego. Este exercício de planejamento de capacidade é um passo importante para determinar quanto o site custará para operar. É difícil estimar antecipadamente quanto tráfego um site receberá, por isso é prática comum planejar capacidade suficiente para satisfazer os níveis de tráfego além das estimativas mais altas (MARKOVIC, 2021;UTOMO, 2020).

Com arquiteturas tradicionais, nas quais as solicitações de página podem exigir atividade em todos os níveis da pilha, essa capacidade precisará se estender por cada camada, geralmente resultando em vários servidores altamente especificados para bancos de dados, servidores de aplicativos, servidores de cache, balanceadores de carga, filas de mensagens e muito mais. Cada uma dessas peças de infraestrutura tem um custo financeiro

associado. Anteriormente, isso poderia incluir o custo das máquinas físicas, mas hoje essas máquinas geralmente são virtualizadas. Seja físico ou virtual, os custos financeiros dessas peças de infraestrutura podem aumentar, com licenças de software, custos de máquina, mão de obra e assim por diante (UTOMO, 2020; ZAMMETTI, 2020).

Além disso, a maioria dos projetos de desenvolvimento web terá mais de um ambiente, o que significa que grande parte dessa infraestrutura precisará ser duplicada para fornecer ambientes adequados de teste, teste e desenvolvimento além do ambiente de produção (UTOMO, 2020; ZAMMETTI, 2020).

JAMstack se beneficiam de uma arquitetura técnica muito mais simples. O ônus de dimensionar um site JAMstack para atender a grandes picos de tráfego normalmente recai sobre a Content Delivery Network (CDN) que está atendendo os ativos do site. Mesmo que não empregássemos os serviços de uma CDN, nosso ambiente de hospedagem ainda seria drasticamente simplificado quando comparado ao cenário mencionado. Quando o processo de acessar o conteúdo e os dados e, em seguida, preencher os templates das páginas é desacoplado das solicitações para essas páginas, significa que as demandas sobre essas partes da infraestrutura não influenciam o número de visitantes do site. Grandes partes da infraestrutura tradicional não precisam ser dimensionadas ou talvez nem precisem existir (UTOMO, 2020; ZAMMETTI, 2020).

O JAMstack reduz drasticamente os custos financeiros de construção e manutenção de sites e aplicativos.

4.23 Eficiência da Equipe

O tamanho e a complexidade da arquitetura de um projeto são diretamente proporcionais à quantidade de pessoas e ao leque de habilidades necessárias para operá-lo. Uma arquitetura simplificada com menos servidores requer menos pessoas e muito menos especialização (MARKOVIC, 2021;UTOMO, 2020).

Tarefas complexas de DevOps são amplamente removidas de projetos com ambientes mais simples. Procedimentos que antes eram demorados,

caros e críticos – como provisionar novos ambientes e configurá-los para replicar fielmente uns aos outros – são substituídos pela manutenção apenas dos ambientes de desenvolvimento local (exigidos com uma abordagem tradicional, de qualquer maneira) e o pipeline de implantação para um serviço de hospedagem estática produtizado ou CDN (MARKOVIC, 2021;UTOMO, 2020).

Essa mudança coloca muito mais poder e controle nas mãos dos desenvolvedores, que possuem um conjunto cada vez mais amplo de habilidades de desenvolvimento web. Isso reduz o custo de contratação de pessoal para um projeto de desenvolvimento web (através de uma demanda reduzida por algumas das habilidades mais exóticas ou historicamente caras) e aumenta a produtividade dos desenvolvedores empregados. Por ter conhecimento prático de uma porção maior da pilha e menos limites de disciplina a serem cruzados, o modelo mental de cada desenvolvedor do projeto pode ser mais completo e, como resultado, cada indivíduo pode ter mais confiança em suas ações e ser mais produtivo (MARKOVIC, 2021;UTOMO, 2020).

4.24 O preço da inovação

Ao fazer alterações em um site, precisamos ter certeza do efeito que nossas alterações podem ter no restante do sistema.

O JAMstack aproveitou APIs e uma arquitetura de serviços bem definidos com camadas de interface estabelecidas, nos movendo em direção a um sistema que abraça o mantra de “pequenos pedaços, unidos livremente”. Esse modelo de acoplamento fraco entre diferentes partes da arquitetura técnica de um site reduz as barreiras à mudança ao longo do tempo. Isso pode ser libertador quando se trata de tomar decisões de design técnico, porque é menos provável que fiquemos presos a um fornecedor ou serviço terceirizado específico, já que temos limites e responsabilidades bem definidos em todo o site.

Isso também dá às equipes de desenvolvimento liberdade para refatorar e desenvolver as partes específicas do site que controlam, com a certeza de que, desde que respeitem a estrutura das interfaces entre as diferentes partes

do site, não comprometerão o projeto mais amplo (MARKOVIC, 2021;UTOMO, 2020).

Enquanto as arquiteturas monolíticas sufocam a capacidade de iteração, com acoplamento rígido e infraestruturas proprietárias, o JAMstack pode quebrar esses limites e permitir que a inovação floresça.

Claro, ainda é possível projetar um site JAMstack de uma maneira que crie um sistema fortemente acoplado ou que tenha muitas interdependências que dificultam a mudança. Mas podemos evitar isso porque a simplificação da infraestrutura necessária leva a uma maior clareza e compreensão das partes constituintes do site (MARKOVIC, 2021;UTOMO, 2020).

Acabamos de discutir como a redução das partes móveis do sistema em tempo de execução leva a uma maior confiança no atendimento ao site. Isso também tem um impacto significativo na redução do custo da inovação. Quando qualquer complexidade pode ser exposta em tempo de construção em vez de em tempo de execução, podemos ver os resultados de nossas modificações com muito mais segurança, permitindo maior experimentação e confiança no que será implantado (MARKOVIC, 2021;UTOMO, 2020).

Da simplificação do sistema a modelos mentais mais ricos e maior capacitação das equipes de desenvolvimento, com o JAMstack, o custo da inovação pode ser reduzido significativamente (MARKOVIC, 2021;UTOMO, 2020).

4.25 Régua

A capacidade da infraestrutura de hospedagem para atender às demandas do público de um site é fundamental para seu sucesso. Falamos sobre o custo associado ao planejamento e provisionamento da infraestrutura de hospedagem em pilhas tradicionais e, em seguida, as demandas muito mais simples ao usar o JAMstack. Agora, vamos ver mais de perto como o JAMstack pode ser dimensionado e por que ele é tão adequado para entregar sites sob cargas de tráfego intenso (MARKOVIC, 2021;UTOMO, 2020).

Mesmo com o planejamento de capacidade mais rigoroso, há momentos em que nossos sites (espero) receberão ainda mais atenção do que planejamos, apesar de toda a nossa contingência e ambição.

O uso da palavra “espero” na frase anterior é um pouco controverso. Sim, queremos que nossos sites sejam bem-sucedidos e alcancem o público mais amplo possível. Sim, queremos poder relatar níveis recordes de visitantes a partir de nossas análises e ter um grande sucesso em nossas mãos. Mas as equipes de infraestrutura temem regularmente que seus sites possam se tornar muito populares. Que, em vez de um nível de demanda saudável e suave, eles recebem grandes picos de tráfego em momentos inesperados. O medo de estar no topo do Hacker News aparece fortemente nas sessões de planejamento de capacidade, embora seja repetidamente alterado para qualquer site ou propriedade de mídia social que possa ser a fonte mais recente de níveis excessivos de tráfego devido a um site se tornar viral (MARKOVIC, 2021;UTOMO, 2020).

O advento dos servidores virtualizados foi um grande avanço para enfrentar esse desafio, com técnicas disponíveis para monitorar e dimensionar farms de servidores virtuais para lidar com picos de tráfego. Mas, novamente, isso adiciona complexidade ao projeto técnico e à manutenção de um site (MARKOVIC, 2021; UTOMO, 2020).

4.26 Imitando estático para escalar

Uma técnica empregada por muitos sites projetados e projetados para lidar com altos níveis de tráfego (tanto sustentado quanto temporal) é adicionar uma camada de cache e talvez também uma CDN à pilha (UTOMO, 2020; ZAMMETTI, 2020).

Esse é o ponto onde o JAMstack se destaca. Sites construídos em pilhas tradicionais precisam gerenciar seu conteúdo criado dinamicamente em suas várias camadas de cache e CDNs. Este é um conjunto complexo e especializado de operações. A complexidade e o custo resultante levaram a uma percepção de que as CDNs são o domínio de grandes sites corporativos com grandes equipes e grandes orçamentos. O conjunto de ferramentas que eles implementam efetivamente pega o que é dinâmico e o propaga para o cache e CDNs como conjuntos de ativos estáticos para que possam ser servidos pré-fabricados sem uma viagem de ida e volta ao servidor de aplicativos.

Em outras palavras, sites dinâmicos adicionam uma camada extra de complexidade apenas para permitir que sejam atendidos com infraestrutura de hospedagem estática para atender à demanda em escala.

Enquanto isso, com o JAMstack já estamos lá. Nossa compilação pode produzir exatamente os tipos de ativos necessários para ir diretamente para a CDN sem a necessidade de introduzir camadas adicionais de complexidade. Este não é o domínio de grandes sites corporativos, mas está ao alcance de qualquer pessoa que use um gerador de site estático e implante seus sites em um dos vários serviços de CDN disponíveis (UTOMO, 2020; ZAMMETTI, 2020).

4.27 Geografia

Além do volume de tráfego que um site pode receber, também devemos considerar a localização geográfica dos visitantes do nosso site. Se os visitantes estiverem situados no lado oposto do mundo onde nossos servidores residem, eles provavelmente terão um desempenho reduzido devido à latência introduzida pela rede.

Uma boa CDN terá nós distribuídos globalmente, para que seu site seja sempre servido a um visitante do servidor mais próximo. Se a arquitetura escolhida for adequada para colocar seu site na CDN, sua capacidade de atender a um público em qualquer lugar do mundo será bastante aprimorada.

4.28 Redundância

Ao projetar arquiteturas da Web, visamos minimizar os pontos únicos de falha (SPoFs). Quando os sites estão sendo servidos diretamente de uma CDN, nossos próprios servidores não atuam mais como um SPoF que poderia impedir um visitante de acessar o site. Além disso, se qualquer nó individual dentro de uma CDN global falhar (em si um evento bastante importante e improvável), o tráfego seria satisfeito por outro nó, em outro lugar da rede (UTOMO, 2020; ZAMMETTI, 2020).

Este é outro exemplo do benefício de colocar distância entre ambientes de construção e ambientes de serviço, que podem continuar a funcionar e servir tráfego mesmo se nossa própria infraestrutura de construção tiver um problema.

Resiliência, redundância e capacidade são as principais vantagens de entregar sites com o JAMstack (DIAZ, 2018).

4.29 Desempenho

O desempenho importa. À medida que a web alcança mais partes do planeta, os desenvolvedores que a constroem devem considerar a variação da confiabilidade da rede e das velocidades de conectividade. As pessoas esperam poder atingir seus objetivos online em uma variedade cada vez maior de contextos e locais. E os tipos de dispositivos que estão sendo usados para acessar a web nunca foram tão diversos em termos de poder de processamento e confiabilidade (DIAZ, 2018).

Desempenho pode ser interpretado como significando muitas coisas. Nos últimos anos, o valor de servir sites rapidamente e alcançar um tempo rápido para a primeira pintura significativa e o tempo para a interação demonstraram ser críticos para a experiência do usuário (UX), retenção do usuário e conversão. Simplificando, tempo é dinheiro. E quanto mais rápido os sites puderem ser servidos, mais valor eles poderão desbloquear (DIAZ, 2018).

Tem havido muitos estudos de caso publicados sobre isso. Podemos encontrar alguns resultados impressionantes da otimização de desempenho da Web listados em “https://wpostats.com/”, alguns mostrando desempenho marginal gerando melhorias impressionantes, como estas:

“COOK aumentou a taxa de conversão em 7% depois de reduzir o tempo médio de carregamento da página em 0,85 segundos. A taxa de rejeição também caiu 7% e as páginas por sessão aumentaram 10%.”

“A reconstrução das páginas do Pinterest para desempenho resultou em uma redução de 40% no tempo de espera, um aumento de 15% no tráfego de SEO e um aumento de 15% na taxa de conversão para inscrição.”

“A BBC viu que eles perdem mais 10% de usuários para cada segundo adicional que leva para carregar o site.”

Enquanto outros apresentam resultados surpreendentes:

“Vendedor de móveis Zitmaxx Wonen reduziu o tempo de carregamento típico para 3 segundos e viu a conversão saltar 50,2%. A receita geral do site móvel também aumentou 98,7%.”

4.30 Onde para focar na otimização Esforços

Por muito tempo, as otimizações de desempenho para sites focavam principalmente no lado do servidor. Pensava-se que a chamada engenharia de back-end era onde a engenharia séria acontecia, enquanto o código de front-end era visto por muitos como um campo menos sofisticado. Menos atenção _ _ foi pago para otimizações de desempenho no frontend, código do lado do cliente.

Essa situação mudou significativamente. A percepção de que as eficiências na arquitetura e transmissão do código frontend podem fazer uma diferença impressionante no desempenho de um site criou uma disciplina importante na qual melhorias mensuráveis podem ser feitas. Muitos frutos de baixo custo proverbiais foram identificados, e hoje há uma parte estabelecida da indústria de desenvolvimento web focada na otimização de front-end.

No frontend, vemos uma atenção incrível sendo dada a coisas como as seguintes (DIAZ, 2018):

- Minimizar o número de solicitações HTTP necessárias para entregar um site interativo (DIAZ, 2018).
- Arquiteturas de página projetadas para evitar bloqueio de renderização ou dependências de fontes externas.
- Uso de técnicas de imagem e visualização que evitam vídeos e imagens pesados
- Esforços para evitar desgaste do layout ou operações lentas de pintura

Com essas e inúmeras outras áreas, o trabalho de otimização do desempenho no navegador ganhou ferramentas, expertise e valorização.

Mas o desempenho da infraestrutura de serviço da Web e de serviço de aplicativos continua sendo fundamental, e é aqui que vemos os esforços para otimizar por meio de iniciativas como as seguintes:

- Adicionando e gerenciando camadas de cache entre recursos comumente solicitados.
- Afinar consultas de banco de dados e projetar estruturas de dados para minimizar gargalos.
- Adição de poder computacional e balanceamento de carga para aumentar a capacidade e proteger o desempenho sob carga.
- Criação de infraestrutura de rede subjacente mais rápida para aumentar o rendimento nas plataformas de hospedagem.

Cada área é importante. Um bom desempenho na web requer um esforço conjunto em todos os níveis da pilha, e um site terá apenas o desempenho de seu elo de menor desempenho na cadeia.

No entanto, criticamente, o JAMstack remove camadas dessa pilha. Isso encurta essa cadeia. Agora, as operações nas quais as equipes costumavam gastar muito tempo e dinheiro para otimizar em um esforço para acelerá-las e torná-las mais confiáveis não existem em tempo de execução (DIAZ, 2018).

Esses níveis e operações que permanecem podem agora receber mais escrutínio do que antes, porque existem menos áreas para competir por atenção e se beneficiar de nossos melhores esforços de otimização (DIAZ, 2018).

No entanto, não há código que possa ser executado mais rápido que o código zero. Esse tipo de simplificação e separação da construção de sites do serviço de sites produz resultados impressionantes.

Considere um ciclo de vida de solicitação típico em uma arquitetura de site relativamente simples, mas dinâmica (DIAZ, 2018):

i. Um navegador faz uma solicitação para uma página.
ii. A solicitação é tratada por um servidor Web que inspeciona a URL solicitada e roteia a solicitação para a parte correta da lógica interna para determinar como ela deve ser atendida.
iii. . O servidor da Web passa a solicitação para um servidor de aplicativos que mantém a lógica sobre quais modelos devem ser combinados com dados de várias fontes.
iv. O servidor de aplicativos solicita dados de um banco de dados (e talvez de sistemas externos) e os processa em uma resposta.

v. A resposta é passada de volta do servidor de aplicativos para o servidor da Web e, em seguida, para o navegador, onde pode ser exibida ao usuário.

Em sistemas como o que acabamos de descrever, com backend dinâmico, tornou-se comum a introdução de camadas adicionais para ajudar a melhorar o desempenho. Podemos ver a introdução de uma camada de cache entre o servidor web e o servidor de aplicativos, ou mesmo entre o servidor de aplicativos e os bancos de dados. Muitas dessas camadas realmente introduzem operações que se comportam como se aquela parte do sistema fosse estática, mas precisam ser gerenciadas e atualizadas pelo sistema ao longo da operação.

Agora vamos considerar o mesmo cenário para o JAMstack no qual a compilação das visualizações de página não é realizada sob demanda, mas antecipadamente em um único processo de compilação (DIAZ, 2018):

- Um navegador faz uma solicitação para uma página
- Um CDN corresponde essa solicitação a uma resposta pronta e a retorna ao navegador, onde pode ser exibida ao usuário

Imediatamente podemos ver que muito menos acontece para satisfazer cada solicitação. Há menos pontos de falha, menos distância lógica a percorrer e menos sistemas interagindo para cada solicitação. O resultado é um desempenho aprimorado na infraestrutura de hospedagem por padrão. Os ativos são entregues ao frontend o mais rápido possível porque estão prontos para serem atendidos e já estão o mais próximo possível do usuário graças ao uso do CDN.

4.31 Execução apenas quando necessário

Essa pilha de solicitação/resposta significativamente menor é possível porque as páginas não estão sendo montadas por solicitação; em vez disso, o CDN foi preparado com todas as visualizações de página que sabemos que nosso site precisará (DIAZ, 2018).

Mais uma vez, estamos nos beneficiando de dissociar a geração e a população de páginas do momento em que são necessárias. Eles estão prontos para serem servidos no momento em que são solicitados.

Além disso, podemos ser avisados com antecedência sobre quaisquer problemas que possam surgir durante a geração dessas páginas. Caso esse problema ocorra em um site onde as páginas estão sendo geradas e veiculadas sob demanda, precisaríamos retornar um erro ao usuário.

Não é assim em um site JAMstack para o qual a geração de páginas ocorre antecipadamente durante uma implantação. Lá, se uma operação de geração de página falhar, o resultado será uma implantação com falha que nunca será transmitida ao usuário. Em vez disso, a falha pode ser relatada e corrigida em tempo hábil. Sem que nenhum usuário seja afetado ou ciente (DIAZ, 2018).

4.32 Segurança

A complexidade reduzida do JAMstack oferece outra vantagem: um perfil de segurança muito aprimorado para nossos sites (LINDLEY, 2017).

Ao considerar a segurança, o termo área de superfície é frequentemente usado para descrever a quantidade de código, de infraestrutura e a escala da arquitetura lógica em jogo em um sistema. Menos peças de infraestrutura significam menos coisas para atacar e menos coisas nas quais precisamos nos esforçar para corrigir e proteger. Reduzir a área de superfície é uma boa estratégia para melhorar a segurança e minimizar as vias de ataque (LINDLEY, 2017).

Como já aprendemos, o JAMstack se beneficia de uma pilha menor e simplificada quando comparada às arquiteturas tradicionais com seus vários servidores e bancos de dados, cada um precisando da capacidade de interagir uns com os outros e trocar dados. Ao remover essas camadas, também removemos os pontos em que elas interagem ou trocam dados.

As oportunidades para comprometer o sistema são reduzidas. A segurança é melhorada.

Além de melhorar a segurança reduzindo a área de superfície, os sites JAMstack desfrutam de outro benefício. As peças de infraestrutura que

permanecem envolvidas no atendimento aos nossos sites não incluem código lógico a ser executado em nossos servidores no momento da solicitação. Na medida do possível, evitamos que os servidores executem códigos ou efetuem alterações em nosso sistema (LINDLEY, 2017).

Em sistemas somente leitura, não apenas as considerações de escala e desempenho são aprimoradas, mas nosso perfil de segurança é aprimorado.

Podemos voltar à discussão dos sites Wordpress como uma comparação útil. Um site Wordpress combina um banco de dados, camadas de lógica escritas em PHP, modelos para apresentação e uma interface de usuário para configurar, preencher e gerenciar o site. Esta interface de usuário é acessada pela web via HTTP, exigindo que um site Wordpress seja capaz de aceitar solicitações HTTP POST e consumir e analisar os dados enviados a ele nessas solicitações (LINDLEY, 2017).

Isso abre um vetor de ataque popular e muitas vezes bem-sucedido para sites Wordpress. Um esforço considerável foi investido na tentativa de proteger essa rota de ataque, e aderir estritamente ao estabelecimento de boas práticas pode ajudar. Mas isso não pode garantir a segurança total.

O tráfego especulativo e hostil que investiga interfaces de administração do Wordpress mal protegidas é abundante na Internet. É automatizado, prolífico e continua a melhorar em sofisticação à medida que novas vulnerabilidades são descobertas.

Para aqueles que procuram proteger um site Wordpress, é uma batalha difícil vencer de forma permanente e confiante. O JAMstack é diferente. Em vez de começar com uma arquitetura que permite operações de gravação para servidores executarem código e, em seguida, tentar proteger isso mantendo as portas para essas operações bem protegidas, um site JAMstack não possui partes móveis ou portas para proteger. Sua infraestrutura de hospedagem é somente leitura e não é suscetível aos mesmos tipos de ataque (LINDLEY, 2017).

Exceto, estamos trapaceando aqui. Estamos falando de sites JAMstack como se fossem estáticos, tanto na infraestrutura de hospedagem quanto na experiência do usuário. No entanto, também estamos tentando transmitir que os sites JAMstack podem ser tão dinâmicos em experiência quanto muitas outras arquiteturas, então o que dá? (LINDLEY, 2017)

Estamos simplificando um pouco e pulando aspectos de sites de pilha JAM que podem interagir com outros sistemas. O JAMstack certamente inclui todos os tipos de serviços interativos ricos que podemos usar em nossos sites. Falamos de um ecossistema crescente de serviços à nossa disposição, então como conciliar isso no que diz respeito à segurança? (LINDLEY, 2017)

4.33 Terceirizado ou Serviços Abstraídos

Até agora, focamos na infraestrutura que nós, como proprietários de sites, precisaríamos para nos proteger e operar. Mas se quisermos criar experiências dinâmicas, haverá momentos em que nossos sites precisarão interagir com sistemas mais complexos. Existem algumas abordagens úteis disponíveis para minimizar qualquer impacto negativo na segurança. Vamos examinar alguns eles.

4.34 Complexidade para terceirização

Lembrando que o “ A ” no JAMstack significa APIs, não deveria ser surpresa que descrevemos vantagens no uso de APIs para interagir com serviços externos. Ao usar APIs para permitir que nossos sites interajam com um conjunto discreto de serviços, podemos obter acesso a uma ampla variedade de recursos adicionais muito além do que gostaríamos de fornecer a nós mesmos. O mercado de recursos como serviço é amplo e próspero (MARKOVIC, 2021, BILLINGSLEY, 2020).

Ao selecionar cuidadosamente fornecedores especializados em fornecer um recurso específico, podemos tirar proveito de sua experiência de domínio e terceirizar para eles a necessidade de conhecimento especializado de seu funcionamento interno. Quando essas empresas oferecem esses serviços como suas principais capacidades, elas assumem a responsabilidade de manter sua própria infraestrutura da qual seus negócios dependem (MARKOVIC, 2021, BILLINGSLEY, 2020).

Esse modelo também contribui para uma melhor separação lógica de serviços individuais e recursos subjacentes, o que pode ser vantajoso quando

se trata de manter um conjunto de recursos em seu site. O resultado é uma separação clara das preocupações, além das responsabilidades de segurança.

4.35 Abstração e dissociação

Nem todos os recursos podem ser terceirizados e consumidos por meio de APIs. Há algumas coisas que precisamos manter internamente devido ao valor comercial ou porque os serviços externos não são capazes de atender a alguns requisitos mais personalizados.

O JAMstack está bem posicionado para minimizar a implicação de segurança aqui também. Por meio da dissociação da geração de sites por meio da pré-renderização e do atendimento de nossos sites depois de gerados e implantados, distanciamos o acesso público aos nossos sites e os mecanismos que os constroem. Anteriormente, pedimos cautela ao fornecer acesso a sites Wordpress apoiados por um banco de dados, mas é muito provável que algumas partes de nossos sites precisem acessar o conteúdo mantido em um banco de dados. Se o acesso a esse banco de dados for totalmente abstraído da hospedagem e do serviço de nossos sites, e não houver acesso público a esse recurso (porque ele é acessado por meio de um processo de construção totalmente desacoplado), nossa exposição à segurança é significativamente reduzida (MARKOVIC, 2021, BILLINGSLEY, 2020).

JAMstack incorporam estes princípios (MARKOVIC, 2021, BILLINGSLEY, 2020):

- Uma área de superfície mínima com infraestrutura de hospedagem em grande parte somente leitura
- Serviços desacoplados expostos ao ambiente de construção e não ao público
- Um ecossistema de serviços externos seguros e operados de forma independente

Todos esses princípios melhoram a segurança e reduzem a complexidade que precisamos gerenciar e manter.

4.36 Para o Desenvolvedor; Para o projeto; Para a vitória

Ao longo deste capítulo, falamos sobre as vantagens que advêm da complexidade reduzida. Falamos sobre como a remoção da complexidade comum pode reduzir os custos do projeto, melhorar a capacidade de dimensionar e atender a grandes volumes de tráfego e melhorar a segurança (MARKOVIC, 2021, BILLINGSLEY, 2020).

Tudo isso é uma boa notícia para um projeto. Eles são bons para o resultado final. Eles melhoram a probabilidade de um projeto ser aprovado para prosseguir e suas chances de sucesso prolongado. Mas se isso vier ao custo de uma experiência de desenvolvimento sólida, temos algumas ponderações a fazer (MARKOVIC, 2021, BILLINGSLEY, 2020).

A experiência do desenvolvedor é um importante fator de sucesso. A capacidade de um desenvolvedor de cumprir efetivamente a promessa do design, da estratégia e da visão muitas vezes pode ser a peça final do quebra-cabeça. Isso é Onde idéias são realizadas.

Uma experiência de desenvolvimento ruim pode ser devastadora para um projeto. Pode impedir o progresso do desenvolvimento e criar implicações de manutenção que prejudicam a saúde de longo prazo de um projeto. Pode tornar mais difícil recrutar e reter as pessoas de que precisamos para entregar um projeto. Pode gerar frustrações com o progresso lento ou baixa confiabilidade. Pode sufocar qualquer capacidade de inovar e tornar um projeto excelente (MARKOVIC, 2021, BILLINGSLEY, 2020).

Uma boa experiência de desenvolvedor pode ajudar a criar alta produtividade, uma equipe feliz e inovação em todos os níveis de um projeto. é algo para se esforçar (MARKOVIC, 2021, BILLINGSLEY, 2020).

O que precisamos é de uma arquitetura e uma abordagem de desenvolvimento que de alguma forma satisfaça esses dois critérios. O JAMstack posso entregar isso para nós (MARKOVIC, 2021, BILLINGSLEY, 2020).

A simplificação que anunciamos tantas vezes neste capítulo não vem à custa da clareza. Não introduz ofuscação ou magia opaca. Em vez disso, ele emprega ferramentas, convenções e abordagens que são populares e cada vez mais disponíveis entre os desenvolvedores da web. Ele abrange fluxos de

trabalho de desenvolvimento projetados para aprimorar a capacidade de um desenvolvedor de ser eficaz e construir coisas em ambientes familiares, mas poderosos. Cria fortes limites lógicos entre sistemas e serviços, criando áreas claras de foco e propriedade. Não precisamos escolher entre uma experiência de desenvolvedor eficaz e uma experiência de usuário eficaz. Com a JAMstack nós posso ter ambos (MARKOVIC, 2021, BILLINGSLEY, 2020).

5 FRAMEWORKS

5.1 GRATSBY

Gatsby JS é um gerador de site estático poderoso, flexível e extremamente rápido para React JS. É um produto de tecnologia web moderna que aproveita todos os recursos de inovações avançadas de código aberto nas comunidades React, NPM e Gatsby (ZAMMETTI, 2020).

Indo além dos nomes, porém, Gatsby é mais do que apenas um SSG. Traz ao desenvolvedor a velocidade de um site gerado estaticamente e também a funcionalidade de um framework sofisticado. Ele ajuda a destruir a percepção de que os geradores de sites estáticos são adequados apenas para sites simplistas, como conteúdo ou marketing, e ajuda a criar sites estáticos que carregam dinamicamente e podem transmitir experiências ricas em recursos e suaves como aplicativos na web, graças ao seu DOM reidratado. Assim, de sites de comércio eletrônico a provedores de funções sem servidor, o Gatsby ajuda a criar aplicativos avançados e experiências na web (ZAMMETTI, 2020).

5.1.1 Melhores recursos do Gatsby JS

- Baseado em React: Uma das melhores características do Gatsby é que sendo baseado em React, ele tem uma curva de aprendizado simples e pode ser facilmente adotado por desenvolvedores em geral. Qualquer coisa que se queira construir, seja um site de comércio eletrônico, um blog ou um site

corporativo, se o desenvolvedor pode construir com React, ele pode construir com Gatsby.

- Velocidade: Gatsby ultrapassa os limites do desenvolvimento web tradicional, permitindo que se exiba experiências semelhantes a aplicativos com um orçamento apertado. Além disso, permite surpreender os visitantes com velocidade incrível e capacidade de resposta incomparável. Combinando a geração de site estático e renderização de página inteligente, o Gatsby oferece experiências suaves e ricas em recursos em escala.
- Carregar dados de qualquer lugar: um dos recursos pelos quais Gatsby é amplamente amado é sua capacidade de carregar dados de qualquer lugar. Seja um sistema de gerenciamento de conteúdo como WordPress, REST, Contentful ou GraphQL API, ou arquivos Markdown.

Gatsby introduziu o conceito único de 'malha de conteúdo'. No blog oficial do Gatsby, Sam Bhagwat, co-criador do Gatsby JS descreve a malha de conteúdo como: *"A estratégia emergente para gerenciamento de conteúdo seleciona as melhores soluções, adaptadas a casos de uso específicos, como comércio eletrônico ou blogs; ele fornece uma estrutura moderna para iteração rápida e gera sites rápidos e prontos para uso."* (ZAMMETTI, 2020)

O Gatsby JS elimina a dor da integração manual e oferece aos desenvolvedores acesso aos melhores serviços, preservando os fluxos de trabalho dos criadores de conteúdo. Assim, o desenvolvedor pode vincular facilmente plataformas de terceiros e lidar com uma variedade de fontes de dados. Dessa forma, a malha de conteúdo se afasta das plataformas monolíticas e tudo-em-um e se aproxima de um modelo coeso em que cada plataforma de terceiros contribui com uma funcionalidade especial para a arquitetura geral, permitindo que cada plataforma faça o que faz de melhor, enquanto o desenvolvedor obtém uma experiência de desenvolvimento uniforme, não importa de onde os dados venham (ZAMMETTI, 2020).

Ótimo desempenho integrado: Os criadores do Gatsby JS claramente tinham o desempenho em seu coração e projetaram o Gatsby para oferecer um

ótimo desempenho imediatamente. O processo de construção do Gatsby é otimizado para oferecer o máximo desempenho do site graças a estratégias como inclinação de ativos críticos e divisão de código eficaz, pré-busca de recursos e muito mais. A divisão de código garante que apenas o código necessário para cada página seja servido, aumentando a velocidade e o desempenho. Além disso, a otimização pesada da imagem e o carregamento lento ajudam a carregar até mesmo as imagens ricas mais rapidamente e mantém a experiência em movimento sem pausa (ZAMMETTI, 2020).

Hospedagem de baixo custo em escala: Como os sites Gatsby não exigem servidores, é possível praticamente hospedar seu site por centavos em um CDN e até mesmo totalmente gratuito no Gatsby Cloud. Ao contrário dos sites renderizados pelo servidor, o Gatsby produz conteúdo estático que é incrivelmente fácil e acessível para hospedar com qualquer provedor. Além disso, ao implantar seu site na borda, o Gatsby aproveita as tecnologias modernas que protegem seus dados das vulnerabilidades tradicionais do lado do servidor, proporcionando uma experiência sem complicações para os visitantes.

Gatsby permite que o desenvolvedor tenha ciclos de desenvolvimento mais curtos: os desenvolvedores preferem trabalhar com Gatsby. Ele reúne o melhor das tecnologias modernas da Web, como JavaScript, Git, CI/CD e APIs para oferecer uma experiência de desenvolvedor sofisticada e intuitiva. Os desenvolvedores passam menos tempo fazendo otimizações, liberando mais tempo para escrever código (ZAMMETTI, 2020).

Extenso ecossistema de código aberto: Gatsby JS tem o direito de se gabar de uma comunidade calorosa de código aberto que torna o desenvolvimento uma alegria. Esta comunidade Gatsby até agora criou mais de 2000 plugins, o que significa que a maior parte do que se deseja construir em seu site pode já estar disponível como uma ferramenta pronta que se pode integrar facilmente ao seu site. Adicione a isso uma fantástica coleção de ganchos de API e os desenvolvedores podem personalizar cada etapa do processo de desenvolvimento, seja adicionando ferramentas de terceiros, integrando dados ou convertendo dados de um formato para outro (ZAMMETTI, 2020).

5.1.2 Vantagens

Gatsby tem muitos benefícios para empresários, profissionais de marketing e desenvolvedores. Independentemente da sua posição na empresa, o desenvolvedor deve encontrar algo para si mesmo (ZAMMETTI, 2020).

- Atualização: Gatsby é uma tecnologia em constante evolução e, portanto, os sites ou aplicativos Gatsby criados resistirão ao teste do tempo.
- Metadados da página: Graças aos componentes do capacete de reação, é possível definir metadados (títulos de página, meta descrições, textos alternativos) para o seu site. Isso ajudará o desenvolvedor a ter uma classificação mais alta nos resultados de pesquisa, porque essas coisas ajudam os mecanismos de pesquisa a entender melhor o conteúdo do seu site.
- Gatsby Cloud: É uma infraestrutura de nuvem personalizada que oferece quatro recursos interessantes que os desenvolvedores adoram:
- Builds Incrementais que tornam os tempos de compilação 1000x mais rápidos:
- Pré-visualizações de CMS em tempo real
- Implantar visualizações
- Relatórios do farol
- Enorme ecossistema

Gatsby oferece acesso a muitos plugins, starters, clichês e pacotes React – tudo isso para impulsionar o desenvolvimento.

- Fluxo de trabalho moderno: Gatsby segue e tira proveito dos mais recentes padrões e tecnologias da Web, como React, GraphQL e Webpack.

- Escalabilidade também conhecida como resistência ao tráfego: Com o Gatsby, o desenvolvedor não precisa se preocupar com altos picos de tráfego – seu site ainda estará funcionando. Além disso, os custos dependem do uso, portanto, o desenvolvedor não pagará por algo (transferência de uso) que não usa.
- Acessibilidade: O Gatsby possui alguns recursos que tornam os sites acessíveis, como roteamento acessível ou aprimoramento progressivo de páginas. Além disso, WebAIM reconhecido Gatsby como o framework mais acessível.

5.1.3 Desvantagens

O desenvolvedor não deve utilizar o Gatsby, se (ZAMMETTI, 2020):

- **Planeja gerar muito conteúdo:** gerar um site estático pode levar até 15 minutos de cada vez para um blog grande. No entanto, o desenvolvedor pode aumentar esse processo em até 1.000 vezes graças ao Gatsby Cloud e ao Incremental Builds.
- **O conteúdo será atualizado com frequência:** porque, ao contrário de sites criados com, por exemplo, WordPress, essas atualizações não serão visíveis imediatamente (o atraso é causado pelo tempo de construção)
- **O desenvolvedor quer construir uma loja virtual de tamanho corporativo:** mais uma vez, os tempos de construção vêm ao palco. O problema é que quanto mais conteúdo (ou produtos), mais tempo levará para atualizar as alterações. No entanto, como já mencionado, existe uma solução chamada Builds Incrementais. O desenvolvedor deve lembrar-se de duas coisas –o desenvolvedor precisa pagar por isso, e as atualizações ainda não serão visíveis instantaneamente.

5.2 HUGO

Hugo é um gerador de sites estáticos escrito em Go para ajudar os desenvolvedores a construir sites rápidos e modernos de uma forma divertida. Ao contrário dos sites dinâmicos que constroem páginas a cada visita, Hugo as constrói apenas quando o conteúdo é criado ou atualizado. Assim, Hugo visa duas coisas – uma experiência de usuário ideal para os visitantes e uma experiência de escrita para os autores.

5.2.1 Prós de Hugo

O Hugo oferece alguns benefícios que variam um pouco dos profissionais de outros frameworks Jamstack. Eles são apresentados abaixo.

- **Desempenho:** Hugo usa um servidor HTTP (hospedado no computador) que precisa de uma fração de memória e CPU para atender sites estáticos em comparação com sites dinâmicos.
- **Velocidade incrível:** o site médio é construído em menos de um segundo (<1ms por página).
- **Gerenciamento de conteúdo sólido: o** Hugo foi desenvolvido com o conteúdo em mente, pois oferece suporte a tipos de conteúdo ilimitados, taxonomias, conteúdo dinâmico orientado por API e muito mais.
- **Mais de 300 temas:** O desenvolvedor não precisa construir um site Hugo do zero, pois pode escolher um dos muitos temas e trabalhar a partir daí.
- **Modelos integrados: os modelos** pré-fabricados oferecidos pelo Hugo facilitam o cuidado e a implementação de coisas como SEO, comentários, análises e outras funções.
- **Multilinguismo:** Hugo oferece suporte completo para i18n (internacionalização) para sites multilíngues que são tão fáceis de desenvolver quanto sites de um único idioma.
- **Códigos de acesso:** Hugo oferece códigos de acesso que se traduzem em beleza e flexibilidade no uso do Markdown.

5.2.2 Contras de Hugo

As desvantagens do Hugo são apresentadas abaixo.

- **Curva de aprendizado íngreme:** Familiarizar -se com a sintaxe do Go e a estrutura do projeto pode levar algum tempo. E o desenvolvedor precisa estar confortável com a linguagem Go para usar o Hugo livremente.
- **Falta de sistema de plugins:** Se o desenvolvedor encontrar algumas limitações definidas pelo Hugo, poderá ter problemas para superá-las.

5.3 Next.js

O Next.js oferece suporte a desenvolvedores com suporte a CSS integrado, suporte a TypeScript, otimização automática de imagens ou geração estática incremental.

5.3.1 Desvantagens

Embora o número de prós do Next.js seja muito maior do que o número de contras, vamos ser justos e apontá-los.

- **Desenvolvimento e gerenciamento** – Next.js oferece grande flexibilidade, mas requer gerenciamento contínuo. É uma tarefa para uma pessoa com conhecimento adequado, porém, não necessariamente um desenvolvedor.
- **Custo contínuo** – liberdade de front-end dada pelo Next.js significa a necessidade de construir a camada de front-end a partir do zero. Fazer alterações nele de tempos em tempos exigirá um desenvolvedor.
- **Falta de gerenciador de estado embutido** – instalar Redux, MobX ou qualquer outra coisa é necessário se o desenvolvedor quiser ter um gerenciador de estado.
- **plug-ins** – ao contrário do Gatsby, o Next.js não oferece muitos plug-ins.

Next.js é outra estrutura JavaScript que permite que o desenvolvedor crie sites estáticos e aplicativos da Web de alto desempenho e amigáveis usando React.

Mas o Next.js é mais do que apenas outro gerador de site estático. Graças à Otimização Estática Automática, os desenvolvedores podem criar aplicativos híbridos contendo páginas renderizadas pelo servidor e renderizadas estaticamente. Em outras palavras, não há mais necessidade de escolher entre “estático” e “dinâmico”.

Next.js compartilha benefícios com outros geradores de sites estáticos.

5.3.2 Benefícios

- **Tempo mais rápido para o mercado:** O Next.js ajuda o desenvolvedor a criar MVP rapidamente graças aos componentes e compatibilidade prontos para uso. Além disso, economiza tempo e dinheiro, pois o desenvolvedor pode obter feedback de usuários reais rapidamente e fazer alterações em seu produto de acordo.
- **Totalmente omnicanal:** Pode converter visitantes móveis e de desktop, pois sites e aplicativos criados com o NextJS podem ser acessados de qualquer dispositivo. Também, o desenvolvedor pode usar diferentes vendas canais.
- **Baseado em componentes biblioteca:** O Next.js é construído sobre o React, o que facilita o dimensionamento de sites para cima e para baixo. Na verdade, construir um site é mais como brincar com blocos LEGO.
- **Excelente Apoio, suporte:** A popularidade do Next.js está crescendo constantemente, assim como o número de desenvolvedores e colaboradores. Portanto, não será difícil encontrar um desenvolvedor Next.js sem escrever tudo do zero ou encontrar uma solução para alguns problemas relacionados ao código.
- **Imagem padrão otimização:** Next.js otimiza imagens por padrão e as converte em formatos de imagem modernos como WebP. Além disso, gera tamanhos menores de imagens para serem exibidas em dispositivos móveis.

- **Desenvolvimento vantagens:** O Next.js oferece suporte a desenvolvedores com suporte a CSS integrado, suporte a TypeScript, otimização automática de imagens ou geração estática incremental.

6 TECNOLOGIAS

6.1 HTML e CSS

HyperText Markup Language (HTML) é uma linguagem de marcação (em vez de uma linguagem de programação) para construir documentos estáticos, descrevendo a estrutura do documento. Hoje, é o padrão para a criação de páginas da web e, portanto, principalmente interpretado em um navegador. Para definir a aparência visual (estilo) dos elementos HTML, normalmente o CSS é, e deve ser, utilizado.

Como não há como adicionar comportamento dinâmico a um documento HTML sem mais delongas, o HTML permite incorporar código JavaScript no documento para acessar, modificar, criar e excluir elementos no documento.

6.2 Banco de Dados MySQL

Um banco de dados é uma coleção de dados inter-relacionados, representando informações sobre um domínio específico KORTH (1994), isso significa que para ter um banco de dados é necessário ter informações agrupadas que se relacionam e tratam do mesmo assunto. Para conceituar um banco de dados, DATE(1991) explicou que sistema de banco de dados pode ser considerado como uma sala de arquivos eletrônica.

Um SGBD(Sistema Gerenciador de Base de Dados) é um software que possui recursos capazes de manipular informações do banco de dados e interagir com o usuário. Segundo MILANI (2007), MySQL é um SGBD de fácil utilização com excelente desempenho e estabilidade e suporta qualquer plataforma atual, seu uso de memória também é um fator importante podendo conter até 65 TeraBytes de dados. Além disso, o MySQL é um software open-

source isso significa que qualquer usuário pode alterar seu código fonte para atender suas necessidades, utiliza linguagem SQL (Structured Query Language), ele é desenvolvido em C e C++, isso o torna acessível a vários sistemas operacionais como Windows, Linux, Unix, FreeBSD e Mac OS.

6.3 Bootstrap

O bootstrap é uma coleção de ferramentas desenvolvido através das tecnologias HTML, CSS e javascript. O seu objetivo é facilitar o desenvolvimento de aplicações web. Ele é um framework desenvolvido com o código aberto e originalmente criado pelo Twitter. O bootstrap possui uma coleção de funções que implementam elementos normalmente utilizados pelos desenvolvedores Web.

7 CONCLUSÕES

Este trabalho apresenou a arquitetura JAMStack e como o desenvolvedor que a utiliza deve proceder na fase de planejamento. Foram apresentados os pontos fortes desta arquitetura e o porque ela é aconselhável para aplicações que pretendem implementar SEO (Search Engine Optimization). Adicionalmente, as linguagens de programação normalmente utilizadas nos projetos com esta arquitetura foram apresentados e também três frameworks utilizados para implementar estes sistemas foram analisados.

REFERÊNCIAS

MARKOVIC, Dragana; SCEKIC, Milic. Understanding Jamstack and its Perception in Web Development. 2021.

BILLINGSLEY, William. On the need for open teaching on the JamStack. 2020

GHOSHAL, Pallab Kanti. **Prospects and problems of brick industry**. Mittal Publications, 2008.

MAO, Xiaoli. Comparison between symfony, asp. net mvc, and node. js express for web development. 2018.

GHOSHAL,Pallab Kanti. **Prospects and problems of brick industry**. Mittal Publications, 2008.

HOANG, Tri. JAMStack Continuous Integration and Continuous Deployment with CircleCI and Netlify. 2020.

LINDLEY, Cody. Frontend developer handbook 2017. **Frontend masters**, 2017.

DIAZ, Chris. Using static site generators for scholarly publications and open educational resources. **Code4Lib Journal**, n. 42, 2018.

PELTONEN, Severi; MEZZALIRA, Luca; TAIBI, Davide. Motivations, benefits, and issues for adopting micro-frontends: a multivocal literature review. **Information and Software Technology**, v. 136, p. 106571, 2021.

OROSZ, Edit. Modern Web Development with JAMsatck. 2020.

HOANG, Tri. JAMStack Continuous Integration and Continuous Deployment with CircleCI and Netlify. 2020.

UTOMO, Prayudi et al. Building Serverless Website on GitHub Pages. In: **IOP Conference Series: Materials Science and Engineering**. IOP Publishing, 2020. p. 012077.

ATTARDI, Joe. Introduction to Netlify CMS. In: **Using Gatsby and Netlify CMS**. Apress, Berkeley, CA, 2020. p. 1-12.

ZAMMETTI, Frank. Completing the Game. In: **Practical JAMstack**. Apress, Berkeley, CA, 2020. p. 277-299.