Arquiteturas de Aplicações para Dispositivos Móveis

Igor Costa

2018

Arquiteturas de Aplicações para Dispositivos Móveis

Igor Costa

Sumário

Capítulo 1. Introdução

1.1. Evolução dos dispositivos móveis

Dispositivos móveis (*mobile devices* ou *handhelds*) são dispositivos computacionais pequenos o suficiente para segurar e operar com as mãos, dotados de uma tela (output), que pode ser sensível ao toque (touchscreen), e um teclado (input), que pode ser virtual (projetado na tela e operado via toque) ou físico. Estes dispositivos possuem um sistema operacional capaz de executar programas, chamados de aplicações móveis (aplicativos móveis, mobile applications ou ainda mobile apps), podendo se comunicar com outros dispositivos (com ou sem fio) ou acessar a Internet. Aplicações móveis são instaladas no dispositivo de fábrica, baixadas pelo usuário ou através da loja de aplicativos compatível com o sistema operacional.

Dentre as várias características de aplicações móveis podemos citar a necessidade de monitoramento do nível de energia e prevenção de perda de dados, o armazenamento de dados local e remoto, a sincronização de dados com outros sistemas e aplicativos, o projeto da interface que deve considerar o uso com facilidade com apenas uma das mão, sem a necessidade de apoio em uma superfície, entre muitas outras. Nesta apostila, trataremos aplicativos móveis como os aplicativos desenvolvidos para executar apenas em smartphones, tablets e similares, independentemente da plataforma alvo.

A história dos dispositivos móveis começa na telefonia móvel em 1983, com o lançamento do Motorola DynaTAC 8000X, o primeiro telefone realmente móvel e portátil comercializado. O desenvolvimento dos primeiros protótipos começou em 1973, e gerou uma linha de produtos com lançamentos até 1994. Este dispositivo tinha dimensões de 33 x 4,5 x 8,9 cm, pesava 790g e custava o equivalente a quase 10 mil dólares em valores atuais. A carga completa de sua bateria demorava dez horas e a autonomia era de 30 minutos para ligações. Esta geração de celulares evoluiu rapidamente nos anos seguintes com reduções tanto de dimensões e peso quanto preço, levando à sua rápida popularização.

Do ponto de vista dos aplicativos móveis, foco desta apostila, nos interessa mais o lançamento do Newton MessagePad, pela Apple, em 1993. Este dispositivo tinha como características: tela digital monocromática de 336 x 240 pixels, reconhecimento de escrita, teclado virtual, caneta stylus, quase 1MB de memória total e modem opcional de 9600bps. Ele não emplacou, pois era grande (18 x 11 × 2 cm), pesado (0,41 kg) e caro (equivalente a 1.200 dólares atualmente), porém ainda sim é considerado o marco inicial dos dispositivos móveis, pois foi o primeiro dispositivo com um sistema operacional capaz de permitir aplicativos adicionais carregados pelo usuário em seus 140 KB de armazenamento. Os aplicativos eram escritos na linguagem NewtonScript, desenvolvida especialmente para o dispositivo.

A segunda geração de telefonia móvel, ou 2G, chegou nos anos 90 marcando a transição do padrão analógico para o digital, e seu desenvolvimento surgiu da necessidade de poder aumentar o número de ligações simultâneas com a mesma largura de banda, além de integrar outros serviços como envio de mensagens de texto (SMS), mensagens multimídia (MMS) e capacidade de transmissão de dados entre dispositivos de fax e modem. Entre os protocolos desta geração temos GSM, TDMA, CDMA entre outros. Redes 2G ainda são utilizadas no mundo todo, porém estão em processo de desligamento em vários países com o objetivo de reutilizar as radiofrequências para outros protocolos e aplicações, e com previsões de desligamento para até 2020 nos EUA e em países da Europa. Para o Brasil, existem previsões para desligamento até 2022.

A aceitação de mercado veio após o lançamento do Palm (na época parte da U.S. Robotics) Pilot 1000 e 5000, em 1996. Com dimensões de 12 x 8 x 1,8 cm e peso de 160g, uma tela LCD tátil de 160 x 160 pixels, caneta stylus, cartão de memória, memória RAM de até 512 KB e armazenamento de até 12MB. Seu software era capaz de gerenciar agendas, contatos, listas de tarefas e memorando com facilidade e assim que a sua SDK (*software development kit*) foi liberada uma grande variedade de aplicativos de terceiros foi desenvolvida aumentando ainda mais a relevância da plataforma. Nos anos seguintes, dispositivos Palm chegaram a atingir 80% do mercado mundial, mas foram perdendo relevância com as novos

concorrentes (Android) até ser comprada pela HP em 2010, tendo sua linha de produtos encerrada em 2011.

Também por volta de 1996 surgiram os primeiros dispositivos com o sistema operacional Windows CE 1.0 da Microsoft, como o NEC MobilePro 200 e Casio A-10. Inicialmente a plataforma Windows CE teve baixa aceitação de mercado, porém após o lançamento da versão 3.0 e do Pocket PC, em 2000, a plataforma começou a crescer, sendo utilizada em dispositivos como o HP Jornada e o Compaq Ipaq. O Windows CE era otimizado para dispositivos com pouca memória, e o seu núcleo podia ser executado com apenas 1MB. Sua última versão, 8.0, foi lançada em 2013.

Em 1997 foi lançada a primeira versão comercial do Symbian OS, desenvolvido a partir de uma parceria entre a Nokia, Ericsson, Motorola e Psion formalizada em 1998 através da criação da empresa Symbian Ltd. Este sistema foi pioneiro no uso em smartphones, usava arquitetura ARM, e foi o sistema mais popular entre smartphones até 2010, alcançando 67% de participação de mercado em 2006. Apesar de dominar o mercado, ele era difícil de programar devido à complexidade da sua linguagem de programação nativa (OPL) e do próprio sistema operacional, além dos altos preços de suas IDEs e SDKs, o que desencoraja aplicativos de terceiros. Este problema foi amenizado em versões posteriores através do framework Qt, incluído a partir da versão 3 do Symbian.

Em 2001 entrou em operação comercial no Japão a primeira rede 3G, que era um upgrade dos protocolos 2G para melhorar a velocidade da transmissão de dados. Entre os protocolos dessa geração temos: W-CDMA, HSPA+, HSDPA, CDMA2000 e EDGE. A velocidade de transmissão dependia do protocolo, podendo chegar a 21,6 Mbits/s com HSPA+. Este protocolos trouxeram a possibilidade de usar serviços de localização (como GPS), TV móvel, telemedicina, videoconferência e vídeo sobre demanda.

O primeiro iPhone foi lançado em 2007, após três anos de desenvolvimento e um custo estimado de 150 milhões de dólares, com tela multi-touch, wi-fi, câmera, GPS, com opções de 4GB e 8GB, com preços de 499 e 599 dólares. A AppStore, loja de aplicativos, foi introduzida nas versões 3GS e 4, este último marca a

mudança de nome do sistema operacional iPhone OS para iOS (na versão 4.0). A introdução de uma loja de aplicativos foi um marco que mudou o foco do dispositivo para as aplicações e desenvolvedores, mudança que as plataformas dominantes da época, como Symbian e BlackBerry, não foram capazes de incorporar a tempo.

O DynaBook, conceito criado entre as décadas de 60 e 70 por Alan Kay, foi a primeira proposta de computador estilo tablet. Ele tinha a proposta de levar a computação pessoal às crianças de todas as idades, que era o público alvo, com o objetivo de complementar a educação infantil. As ideias de Alan Key sobre o uso de computadores pessoais na educação de crianças influenciou Steve Jobs e Bill Gates a criarem novas ideias e produtos para dispositivos móveis, apesar de que Alan ainda acredita que o seu conceito ainda não foi totalmente aplicado. Uma das características originalmente listadas era uma bateria com carga quase eterna.

Entre 1998 e 2001 surgiram os primeiros tablets, como conhecemos hoje, com destaque para o Fujitsu Stylistic 2300, considerado o primeiro tablet colorido, que tinha resolução de 800 x 600 pixels, processador x86, 32MB de RAM, 4GB de armazenamento em disco e com desempenho comparável a um laptop. Como sistema operacional, poderia vir tanto com Windows NT quanto com Windows 98. Sua bateria tinha duração aproximada de 5h, suas dimensões eram de 28 x 19 x 4 cm e seu peso de 1.8Kg.

A Apple é frequentemente reconhecida por ter criado uma nova classe de consumidores com o lançamento do iPad, em 2010, que moldou o mercado comercial de tablets nos anos seguintes. A primeira geração do iPad tinha uma tela touchscreen de 9,7" com resolução de 1024 x 768 pixels, 256MB de RAM e tinha versões com somente wi-fi ou com wi-fi e 3G. Inicialmente lançado com uma versão customizada do iPhone OS 3.2 para tablets, ao final de 2010 recebeu a atualização para o iOS 4.2 e ficou em paridade de funcionalidades com o iPhone.

A quarta geração da telefonia móvel, 4G, entrou em operação comercial em 2009 na Noruega e Suécia, com o protocolo LTE. Redes 4G foram especificadas para trabalhar com velocidades máximas teóricas de 100 Mbit/s (*high mobility*) ou 1 Gbit/s (*low mobility*). Atualmente as redes 4G são usadas por um quarto das

conexões mundiais e sua cobertura alcança dois terços da população mundial. Apesar de estarem disponíveis na maioria dos países, sua maior adoção está concentrada em poucos países como EUA e China. A expectativa é alcançar 40% da população até 2020.

1.2. Dispositivos móveis robustos

O termo “robusto” (*robust* ou *rugged*) indica que o dispositivo foi projetado e construído para operar de forma confiável em ambientes e condições adversas. Tais condições incluem: resistência à água (*waterproof*), poeira (*dustproof*), queda (drop-proof), choques e vibrações (shockproof). Dispositivo normais porém envoltos em uma proteção externa, também designados pelo termo “robustecido”, trazem um grau de proteção menor pois seus componentes internos ainda são frágeis. Em geral tais dispositivos apresentam certa proteção à água e queda.

Para serem considerados robustos, os dispositivos passam por testes de resistência padronizados na indústria pelos testes MIL-STD e pela escala de proteção contra entrada IP Code (*International Protection Marking* ou *Ingress Protection Marking*). Os testes MIL-STD incluem 24 testes de laboratório que garantem que o dispositivo pode lidar com: baixa pressão em grandes altitudes, exposição a variações de temperatura (altas e baixas, além de choques térmicos), chuva, umidade, areia, poeira, vazamentos, choques e vibrações. A escala IP Code, mede a proteção contra poeira e líquidos, usando uma classificação de dois dígitos (por exemplo, a classificação IP65 garante proteção completa contra poeira e contra jatos de água).

A classificação IP Code, norma IEC 60529, visa padronizar o grau de proteção do dispositivo de maneira objetiva para evitar classificações genéricas, utilizadas pelas estratégias de marketing dos fabricantes, como “à prova d’água”. Os dígitos que compõem este código indicam a conformidade com as condições especificadas. No caso do primeiro dígito (cujo valor varia entre 0 a 6), o valor 0 indica que não há proteção contra poeira, o valor 3 indica proteção contra objetos

sólidos com 2,5 mm de diâmetro ou mais, e o valor 6 (grau máximo) indica que o dispositivo é à prova de poeira. Para o segundo dígito (cujo valor varia entre 0 e 9k), o valor 0 indica que não há proteção contra o meio líquido, o valor 7 indica proteção contra imersão temporária em água de até 1 metro por 30 minutos, o valor 8 indica proteção contra imersão contínua, e o valor 9k indica proteção contra jatos de vapor e alta pressão.

A maioria dos dispositivos móveis robustos não possuem partes móveis internas, como coolers e HDDs (*hard disc drive*), substituídos por dissipação energética passiva e SSDs (solid-state drive) que são mais resistentes a choques físicos. Também são comuns reforços, normalmente de alumínio, nos componentes internos e placas para evitar flexionamento além de amortecedores na carcaça externa (case) para reduzir danos internos em caso de queda. As telas sensíveis ao toque utilizam vidro quimicamente reforçado para proteção contra arranhões e rachaduras. E alguns dispositivos são equipados com aquecedores internos para permitir a utilização em ambientes de frio extremo.

Os testes de queda e choque envolvem oito procedimentos diferentes que exercitam choque ou impacto de diferentes maneiras. Um dos mais citados, chamado de “teste de queda em trânsito”, exige que o dispositivo sobreviva a um total de 26 quedas de uma altura de 122 cm, em cada face, borda e canto, sobre uma superfície dura, tal como concreto. Os testes de vibração simulam o transporte em uma pessoa ou veículo através de agitadores de laboratório, com variações na forma de onda, frequência e intensidade das vibrações, de acordo com o tipo de dispositivo e ambiente a serem testado. Os teste de resistência à areia e poeira analisam a penetração de pequenas partículas de poeira, como farinha e areia, que são projetadas sobre o dispositivo em velocidades elevadas, variando a intensidade do vento e da temperatura por várias horas, enquanto o dispositivo é girado.

Nos testes de resistência contra líquidos, o dispositivo é exposto a poderosos jatos de água, vindos de diversas direções; a duração do teste, assim como o volume e a pressão da água, variam de acordo com a classificação do dispositivo. Nos testes de temperaturas extremas, o aparelho é exposto ao calor ou frio

extremos, enquanto ele está desligado (armazenamento), sendo ligado e utilizado (operação), e sendo utilizado em temperaturas operacionais normais, depois de ter sido exposto a temperaturas de armazenamento mais altas (espera tática até a operação). Nos testes de umidade, os dispositivos são expostos ao calor tropical com umidade acima de 90%, durante vários dias; em geral algumas das mesmas características que resultam na resistência a líquidos também previnem danos causados pela alta umidade.

Um exemplo de dispositivo móvel robusto é o Cat S31 que, com certificação IP68, é a prova de água até 1,2m por 35 minutos e impermeável ao pó, além de ser aprovado pela Mil-Spec 810G, o que lhe garante resistência a choques e quedas de até 1,8m. Suporta temperaturas de -30°C a 55°C e tem uma tela que, além de resistente a riscos, permite a utilização sob a luz direta do sol ou mesmo molhado.

1.3. Futuro dos dispositivos móveis

O aumento de poder de processamento dos dispositivos móveis trouxe consigo uma maior demanda energética, o que levou ao aumento da capacidade das baterias, que por outro lado foi limitada pela densidade das tecnologias atuais (íon-lítio). Entretanto, pesquisas recentes apontam novas tecnologias que nos próximos anos poderão suprir essa crescente demanda energética. Baterias de nanofios de ouro (gold nanowire batteries) são capazes de serem recarregadas mais de 200 mil vezes em poucos meses sem demonstrar sinais de degradação. Baterias de estado sólido (solid state lithium-ion), que operam no nível de supercapacitores, são capazes de carregar ou descarregar em apenas sete minutos. Várias tecnologias, como grafeno, estão sendo testadas e desenvolvidas, seja para reduzir o tempo de carregamento, para aumentar a densidade ou para aumentar a vida útil das baterias.

Uso de senhas existem para autenticação é parte da rotina de usuários de computadores há muitos anos, mas atualmente estima-se que o consumidor médio utiliza 25 ou mais sites e aplicativos que exigem acesso por senha. Criar uma senha

segura exige mais de uma dezena de caracteres entre letras, números e pontuação. Para atenuar este problema, já temos alternativas baseadas em biometria disponíveis no mercado, como impressão digital, presente em muitos laptops e smartphones, e reconhecimento facial, que recentemente voltou à tona com o iPhone X e sua funcionalidade de desbloqueio biométrico rápido e adaptativo. Entretanto, nenhum destes métodos funciona de forma confiável em todos os cenários (variações de iluminação, barulho, entre outros). Provavelmente no futuro vários métodos serão utilizados simultaneamente em uma abordagem multifator, onde várias técnicas inclusive de análise comportamental serão utilizadas de forma complementar para identificar o usuário de forma segura.

Fabricantes de smartphones prometem telas sem botões ou bordas para interromper seu design há alguns anos, mas ainda não levaram este conceito ao mercado. É comum vermos um atributo que indica a proporção da tela em relação ao corpo do dispositivo, como por exemplo de 82,9% para o iPhone X. Fabricantes como Vivo e Samsung já demonstraram conceitos, e até registros de patentes, incluindo anúncios de atualizações do Android para otimizar a utilização de telas com designs não convencionais como o do Essential Phone.

Outra promessa ainda não realizada são os smartphones modulares, ou seja, dispositivos que possam ter partes ou peças trocadas individualmente. Entre as várias motivações estão reduzir o lixo eletrônico gerado, aumentar a vida útil dos aparelhos e permitir customizações de configurações para cada perfil de usuário. Iniciativas como Phonebloks trabalham na conscientização de indivíduos e empresas através de vídeos conceituais e palestras, enquanto alguns fabricantes já apresentaram protótipos e alguns modelos já chegaram ao mercado. Um dos projetos mais promissores era o Project Ara, do Google, que permitia trocar praticamente todos os componentes individualmente do dispositivo, foi suspenso indefinidamente em 2016. Outras abordagens mais simples, como o Motorola Moto Z, Essential Phone e LG G5, chegaram ao mercado com a opção de acoplar acessórios extras, como câmeras mais poderosas, baterias e caixas de som. Recente o PuzzlePhone trouxe uma abordagem intermediária, composta por três partes: cérebro (CPU, GPU, RAM, câmeras), estrutura (tela touchscreen e corpo do

dispositivo) e coração (bateria), que podem ser trocadas conforme a necessidade do usuário.

Dispositivos flexíveis apresentam a possibilidade de serem moldados em superfícies curvas e que sejam no futuro talvez até enrolados como um papel. A construção destes dispositivos apresenta diversos desafios como o de tornar a uma tela plana flexível e não interromper as conexões entre os componentes, que também precisam ser flexíveis. Avanços tecnológicos em telas flexíveis já estão no mercado atualmente, como televisões curvas, smartphones com telas que invadem suas bordas (Samsung S7 Edge). Alguns protótipos como o ReFlex incluem sensores para detectar e responder à curvatura atual da tela. A Apple possui patentes para diversos dispositivos flexíveis, como iPads flexíveis que poderiam ser enrolados como um papel, mas ainda sem protótipos divulgados ou previsões de lançamento de produtos.

O mercado dispositivos vestíveis, ou *wearable devices*, vem crescendo em ritmo regular, principalmente nos segmentos de moda e atletas amadores, com o uso de pulseiras fitness. Entretanto as aplicações são ilimitadas, como lentes de contato com assistentes pessoais, calçados que usam o movimento do corpo para carregar o seu smartphone, brincos que monitoram batimentos cardíacos e temperatura corporal, anéis para autenticação de usuário, entre muitas outras. Algumas dessas aplicações ainda estão em fase conceitual, e algumas já estão chegando ao mercado, como é o caso do Token, um anel que permite realizar pagamentos, abrir e fechar portas, logar em websites e até mesmo ligar um automóvel.

Realidade aumentada é uma técnica que sobrepõe objetos virtuais em um ambiente do mundo real, onde usuários interagem com o mundo real, através uma combinação de câmeras e telas, enquanto conteúdo digital é adicionado a ele. Esta técnica está sendo utilizada em jogos, museus, aplicativos educacionais, sistemas especialistas, entre outros. Dispositivos móveis têm se tornado cada vez mais poderosos, o que permitiu que soluções cada vez mais complexas pudessem ser aproveitadas. Além de smartphones, temos óculos como Google Glass que foi

interrompido em 2015, e mais recentemente com o Vaunt da Intel. Ambos têm a proposta de aumentarem a imersão, pois liberam as mãos do usuário para interagir com objetos do mundo real.

Projetos como OmniTouch e Microsoft HoloLens são capazes de criar uma realidade mista, onde além de visualizar objetos virtuais no mundo real (realidade aumentada) é possível interagir com estes objetos como se fossem reais. As aplicações são ilimitadas, desde jogos e entretenimento até prototipação virtual de projetos antes da produção tanto de produtos pequenos quanto de grandes estruturas. A empresa de elevadores Thyssenkrupp já está usando a tecnologia da Microsoft para agilizar a inspeção técnica de elevadores em todo o mundo.

Assistentes virtuais como Apple Siri, Amazon Alexa, Microsoft Cortana e Google Assistant estão cada vez mais inteligentes e com mais funcionalidades, substituindo interfaces tradicionais com o usuário para muitas tarefas, como informações de localização e clima, controle multimídia, compras em e-commerces e suporte de usuários. Tais assistentes podem interagir tanto por voz (utilizando reconhecimento de fala usando processamento de linguagem natural) quanto por texto, sendo este último muitas vezes designado pelo termo chatbot. Os primeiros assistentes surgiram na década de 60 e percorreram um longo caminho até estarem disponíveis nos dispositivos atuais. As previsões são de que as capacidades desses assistentes continuem evoluindo rapidamente, o que ampliará as possibilidades de aplicação.

1.4. Contexto mercadológico

As vendas de dispositivos móveis nos últimos anos superou a venda de PCs, que atualmente estão em declínio. PCs ainda são maioria no ambiente de trabalho, entretanto a diferença no volume de vendas pode indicar uma mudança neste cenário em um futuro breve. Dentre os vários fatores que impulsionaram esta mudança está a portabilidade que trouxe a possibilidade de resolver tarefas rápidas em qualquer lugar, sem depender da mesa de trabalho tradicional.

A maioria dos usuários utiliza múltiplas plataformas (phone, desktop, tablet e outras) diariamente, com padrões diferentes de uso dependendo do horário do dia. Na parte da manhã, o dispositivo mais utilizado são smartphones, na parte da tarde desktops/laptops, e na parte da noite tablets seguidos de perto por smartphones.

Estatísticas recentes mostram que o conteúdo consumido pelo usuário médio em dispositivos móveis, nos EUA, é significativamente maior (51%) em relação ao desktop (42%). E por consequência, caso um negócio não alcance seu público através de pesquisa ou exibição de dispositivos móveis, ou não estiver fornecendo um experiência móvel satisfatória, perderá em comparação com os concorrentes que têm foco em mobile. O crescimento de redes sociais móveis, como Snapchat e Instagram, também impulsionou o aumento no tempo gasto em dispositivos móveis.

No entanto, é necessário interpretar o tempo gasto nos dispositivos com cuidado, pois é mais comum usar a maior parte do tempo em smartphones, verificando e-mails e usando mídias sociais. Uma interpretação simplista dessa divisão leva ao popular mantra do design Mobile-First, enquanto a realidade é que a maioria dos consumidores nos mercados também possui desktops ou tablets que eles costumam usar para obter informações mais detalhadas revisão e compra. Por isso é necessário pensar em estratégia multicanal, onde a experiência do usuário acompanha sua mudança de dispositivo.

Dados recentes mostram que, nos EUA, 90% do tempo do consumidor em dispositivos móveis é gasto em aplicativos. Este é um fator importante na decisão arquitetural para a estratégia móvel de um negócio, já que normalmente as empresas decidem entre desenvolver aplicativos móveis ou sites otimizados para dispositivos móveis devido a restrições orçamentárias. Esse dado de deve ser interpretado com cuidado, como mostra a Figura 22, pois Facebook, mensagens, entretenimento e jogos ocupam a maior tempo gasto, mas o uso do navegador ainda é significativo. O maior impacto dessa informação se dá na estratégia de anúncios móveis para alcançar consumidores usando aplicativos como o Facebook e Instagram.

O Brasil atualmente é o sexto país do mundo em número de usuários de smartphones, aumentando a importância das transações realizadas via dispositivos móveis para o comércio eletrônico. A maior parte das vendas online no país ainda são completadas no desktop, entretanto o m-commerce está crescendo rapidamente, o que reflete a importância de ter os dispositivos móveis em conta para uma estratégia de sucesso. Segundo levantamentos recentes, o número de compras virtuais realizadas em dispositivos móveis cresceu 75% representando, em 2017, 46% do total de transações. Neste mesmo período, o percentual de receita dos lojistas virtuais decorrente de transações mobile subiu de 26% para 35% em 2017.

Do ponto de vista profissional, cada vez mais vemos dispositivos sendo utilizados para a criação ao invés do consumo de conteúdo. Para atender a esta demanda, fabricantes estão integrando teclado acopláveis a tablets, ou seja, reinventando os laptops. O objetivo dessa estratégia é aumentar a produtividade sem reduzir a portabilidade, acelerando a migração de atividades que antes só eram possíveis de serem realizadas em dispositivos maiores. Nesta categoria, denominada tablet “Pro”, temos como exemplos: iPad Pro, Microsoft Surface Pro e Google Pixel C.

1.5. Estratégias B2C e B2B

O termo B2C, ou B to C (*business to consumer*), significa “empresa para consumidor” e indica uma relação de negócios conduzida diretamente entre uma empresa e seus consumidores que são usuários finais dos seus produtos ou serviços. No contexto de e-commerce, se refere a um ambiente ou plataforma virtual onde uma empresa vende seus produtos para seus consumidores, numa relação de apenas consumo. Esse é o modelo de vendas mais popular e conhecido. Recentemente, acompanhando a tendência de crescimento de uso dos dispositivos móveis, empresas B2C estão cada vez mais focadas neste usuários. O sucesso do modelo B2C é definido pela constante evolução nas opiniões, tendências e desejos

dos usuários, o que fez com que as empresas investissem em lançar aplicativos móveis.

O modelo B2C exige que a empresa mantenha um bom relacionamento com seus clientes para garantir que eles retornem. Com o desafio de conquistar de verdade o seu cliente quando você não está cara a cara com ele, negócios B2C precisam gerar uma resposta emocional em seus anúncios para atingir os consumidores. Também exigem investimentos em boas histórias (grandes gatilhos emocionais) através de campanhas de marketing, redes sociais (para o posicionamento, influência e credibilidade) e espaços de diálogo com o consumidor, como ferramentas digitais de feedback, sistemas de recomendação, programas de fidelidade e personalização do atendimento.

O modelo B2B, ou B to B (*business to business*), também chamado de "empresa para empresa", se refere ao tipo de relação que existe entre negócios, como entre um fabricante e um distribuidor, ou entre um atacadista e um varejista. É conduzido entre empresas, sem participação de consumidores individuais. No contexto de e-commerce, se refere a um ambiente onde uma empresa (indústria, distribuidor, importador ou revenda) comercializa seus produtos com outras empresa, numa relação que pode ser de natureza de revenda, transformação ou consumo.

Uma típica cadeia de suprimentos envolve muitas transações B2B, como compras de matérias-primas, componentes e produtos, e por isso requer planejamento para ser bem-sucedida. É essencial um bom gerenciamento do relacionamento comercial, que tipicamente envolve interação profissional anterior à venda, como presença em feiras e congressos para divulgar seus produtos e serviços. Neste contexto, a internet potencializou a capacidade de divulgação de produtos e serviços de negócios B2B, chegando inclusive ao e-commerce onde plataformas de cotação de fornecedores cumprem um importante papel na redução de custos.

O que caracteriza o tipo de relação não é o produto em si, mas a atividade fim à qual essa mercadoria é destinada. Caso a atividade seja de transformação

(indústria vende produtos para outras indústrias) ou revenda (indústria vende para distribuidor ou revenda; distribuidor vende para revenda), o modelo de negócio será B2B. Caso a atividade seja de revenda (indústria, distribuidor ou atacado vende para profissional liberal) ou consumo (distribuidor ou atacado vende para consumidor final; indústria vende para consumidor final), o modelo de negócios será B2C. Uma estratégia não exclui a outra, e muitas empresas praticam mais de uma modalidade ao mesmo tempo, inclusive formatos mistos de negócios. Em muitos casos, quando falamos de e-commerce, o ambiente de comercialização e autoatendimento na loja virtual permanece o mesmo, e o que muda são as políticas de preço, quantidade e condições de pagamento que podem receber configurações customizadas para cada público.

B2C e B2B são as estratégias mais comuns, entretanto vale citar outras estratégias. Consumer to consumer (C2C): engloba todas as transações eletrônicas bens ou serviços efetuadas entre consumidores. Geralmente estas trocas são realizadas (intermediação) através de uma terceira entidade, que disponibiliza a plataforma onde se realizam as transações. Consumer to business (C2B): existe uma inversão completa do sentido tradicional da troca de bens. Este tipo de comércio eletrônico é muito frequente em projetos baseados em crowdsourcing. Um número de indivíduos coloca os seus serviços ou produtos à disposição para serem comprados por empresas que procuram esse tipo de bem.

Business to administration (B2A) ou B2G (business to government) engloba todas as transações realizadas on-line entre as empresas e a administração pública. Esta é uma área que envolve uma grande quantidade e diversidade de serviços, designadamente nas áreas fiscal, da segurança social, do emprego, dos registos e notariado, etc. O tipo de serviços tem vindo a aumentar consideravelmente nos últimos anos com os investimentos feitos em e-government. Consumer-to-Administration (C2A): O modelo Consumer-to-Administration engloba todas as transações eletrônicas efetuadas entre os indivíduos e a Administração Pública.

1.6. Modelos comerciais de aplicações

A criação de uma aplicação móvel com fins comerciais exige a escolha da estratégia de venda, que irá definir as formas e os canais utilizados para levar produtos e serviços até os consumidores. Esta estratégia denominada modelo comercial – ou modelo de negócios, monetização – tem o objetivo de rentabilizar a aplicação. Os modelos passíveis de serem implementados são diversos, entretanto nem sempre é fácil a escolha. Entre as opções mais comuns atualmente, temos: publicidade, venda única, vendas parciais de serviços/funcionalidades que podem ser adquiridos dentro da aplicação (in-app purchases), a subscrição, as comissões de transação e de referência.

A publicidade é um dos modelos comerciais mais populares devido à facilidade de implementação e a aceitação generalizada pelos usuários. Neste modelo de negócio, o usuário não paga por nada no aplicativo: nem para adquiri-lo, nem para adquirir algum produto/serviço dentro dele. Neste caso, os desenvolvedores da aplicação ganham dinheiro dos anunciantes que exibem anúncios dentro do seu aplicativo. Existem várias empresas que intermediam anunciantes e desenvolvedores, como por exemplo a Google (Adsense, AdMob) e a Apple (iAds). Os anúncios podem ser exibidos em diversas partes do aplicativo. A receita pode ser tanto por visualizações (ou seja, cada vez que um usuário visualiza um anúncio dentro do aplicativo) quanto por cliques em anúncios.

No modelo de venda única, o usuário realiza uma vez um único pagamento no momento de adquirir o aplicativo. Este modelo de negócio é muito comum quando se considera a rentabilização de aplicações móveis, visto que o modelo de publicidade exige um grande número de usuários para gerar receita significativa. No entanto, a escolha deste modelo afeta de forma significativa o número de downloads das aplicações móveis, devido à elevada concorrência com aplicações gratuitas que oferecem serviços semelhantes. Frequentemente, os usuários tendem a ser sensíveis ao preço, favorecendo as opções gratuitas quando disponível.

Vendas parciais dentro do aplicativo (*in-app purchases*) são um compromisso entre aplicativos gratuitos e pagos, devido a tendência natural dos utilizadores preferirem fazer o download de aplicativos grátis, por parte dos produtores também existe uma predisposição para disponibilizar as apps como grátis, permitindo depois a compra de funcionalidades adicionais como, por exemplo, a opção de remover a publicidade exibida, de liberar funcionalidades exclusivas, ou de itens especiais e dinheiro virtual em jogos. Atualmente este modelo responde por 71% da receita de aplicativos móveis, com previsões de crescimento. Este método obteve grande destaque em jogos que eram disponibilizados de forma gratuita, para que o usuário se viciasse, e então dentro do jogo eram oferecidos serviços in-app para lhe dar vantagens durante o jogo. Este modelo não está restrito a aplicativos gratuitos, uma vez que aplicativos com pagamento único também podem oferecer serviços adicionais como uma forma de gerar receita adicional pós-venda.

Outro modelo de comercialização que permite disponibilizar inicialmente a aplicação de forma gratuita é o modelo de subscrição ou assinatura. Neste modelo é disponibilizada gratuitamente uma versão reduzida em funcionalidades de um serviço. Depois, tenta-se convencer os usuários a assinarem o serviço completo através de pagamentos recorrentes (mensal, trimestral, anual, etc.).

Uma outra possibilidade para receita é a de comissionamento por transação e indicação para a venda de produtos e serviços de outras empresas. Neste modelo um aplicativo que facilita uma transação vantajosa para as partes (C2C, B2B ou B2C) pode ganhar uma comissão pela transação realizada; como por exemplo no final de um jogo, um convite para fazer o download de outro jogo semelhante pode trazer uma comissão por indicação para o produtor da app. Outro exemplo são aplicativos de cashback, que devolvem parte do dinheiro gasto em uma loja ao usuário.

O modelo de negócios escolhido não precisa ser limitado a apenas uma das opções existentes, e é comum aplicações utilizarem uma combinação de vários ou até de todos simultaneamente. Caso o aplicativo seja destinado a um público mais amplo, distribuí-lo gratuitamente talvez seja a melhor opção inicial, pois existem

várias opções para monetizar um aplicativo já instalado, como vimos. No entanto, ao decidir entre a mais popular, que é a exibição de anúncios, é preciso garantir que os anúncios colocados no aplicativo não afetem a experiência do usuário, pois esta prática pode afastar o usuário.

Capítulo 2. Plataformas

2.1. Android

Em 2003, a empresa Android Inc foi fundada com o objetivo de desenvolver um sistema operacional para câmeras digitais. Pouco tempo depois, após concluir que o mercado de câmeras era muito pequeno, mudaram o foco para dispositivos móveis, entrando na competição com Symbian e Windows Mobile. A empresa foi adquirida pela Google em 2005, que ainda manteve o projeto em segredo. Um protótipo inicial foi divulgado em 2006 sem touchscreen e com um teclado físico, mas logo após o lançamento do iPhone, o Android teve suas especificações alteradas para suportar telas sensíveis ao toque, que pouco depois se tornarem essenciais. O primeiro smartphone disponível no mercado, HTC Dream, só foi lançado em 2008.

Ao longo de dez anos, 27 versões da API foram lançadas, desde o lançamento do primeiro smartphone. A partir da versão 1.5 do Android, denominado Cupcake, todas as versões seguintes receberam nomes de sobremesas, em ordem alfabética, sendo a versão mais recente, 8, denominada Oreo. Dentre as muitas novidades e melhorias incorporadas ao longo do anos podemos citar o lançamento do Android Market e das notificações na barra de status logo na primeira versão, em 2008. No ano seguinte foram incorporados os conceitos de widgets, pequenas aplicações na tela inicial. As versões 2.2 e 2.3 trouxeram otimizações de desempenho do processamento e uso de memória, e a versão 3.0 trouxe uma interface otimizada para tablets. Em 2012 as interfaces de usuário para smartphones e tablets foram unificadas, e o suporte ao 4k foi incluído. A versão 4.4 incluiu pela primeira vez o Android Runtime (ART), um novo ambiente de execução que substituiu a VM utilizada até então na versão 5. Também na versão 5 temos a adoção do Material Design. A versão 6 trouxe suporte ao USB-C e a permissões individuais pós-instalação do aplicativo. Em 2016 foram introduzidas otimizações que reduziram o tempo de instalação de apps assim como o espaço ocupado. E em

2017, uma alteração na arquitetura do Android, denominada Project Treble, com o objetivo de simplificar atualizações do Android pelos fabricantes de hardware.

Fabricantes de hardware criaram interfaces de usuário personalizadas em cima da interface padrão do Android com o objetivo de diferenciar seus produtos dos concorrentes e atrair mais consumidores. Exemplos de interfaces que divergem da interface padrão do Android são Samsung TouchWiz e HTC Sense. Ao longo dos anos essas customizações sofreram críticas, como excesso de aplicativos não solicitados e que não podem ser removidos (bloatware) e divergências das diretrizes de design do Android como o material design a partir de 2014. Em contrapartida, a Google apresentou alternativas de smartphones com uma experiência mais pura do Android, como a linha de smartphones Nexus, e os programas Google Play edition devices e Android One.

Inicialmente chamada de Android Market, a loja de aplicativos do Android foi renomeada para Google Play em 2012. Em 2016 ela alcançou a marca de 82 milhões de app baixados no ano e em 2017 alcançou um total de 3,5 milhões de aplicativos disponíveis. A loja filtra os aplicativos compatíveis com o dispositivo, analisando tanto o hardware (sensores) como a versão do Android utilizada. Aplicativos pagos, incluindo os modelos de pagamento in-app e assinatura, devem usar o sistema de pagamentos da loja, onde a receita é dividida em 70% para os desenvolvedores e 30% para a Google. No caso do modelo de assinatura a divisão é de 85/15.

O Android Software Development Kit é uma coleção de ferramentas para o desenvolvimento de aplicativos Android. Dentre as várias ferramentas incluídas temos a IDE Android Studio, APIs compatíveis com a versão-alvo do Android desejada (por exemplo API 26 para o Android 8.1), um emulador para dispositivos (smartphones, tablets, TV, dispositivos vestíveis), além de diversas ferramentas de linha de comando, como o adb (gerenciador de dispositivos conectados), avdmanager (gerenciador de dispositivos virtuais), sdkmanager (gerenciador de pacotes do SDK), entre outros.

O Android Studio é atualmente a IDE oficial para desenvolvimento de aplicativos Android. Após ter sido inicialmente anunciado em 2013, sua primeira versão estável, 1.0, foi lançada em 2014. Com a proposta de substituir o Eclipse, a IDE foi baseada no IntelliJ (JetBrains) e trouxe um emulador mais rápido, além de refatorações específicas para Android. Além de Java, o Android Studio também é compatível com Android Wear, Kotlin, C++ e NDK.

Uma das ferramentas de linha de comando incluídas no SDK do Android é o Android Debug Bridge, ou adb, que permite a comunicação com um emulador ou dispositivo Android conectado. Dentre as diversas funcionalidades disponíveis temos a verificação dos dispositivos conectados, instalação e depuração de aplicativos, o envio de comandos para serem executados no dispositivo como cópia de arquivos, captura de tela, gravação de vídeos, entre outros.

Dentre as várias opções disponíveis para a programação de aplicativos Android, podemos destacar três: Java, Kotlin e C++. A primeira é também a mais popular, pois acompanha o Android desde suas primeiras versões. Atualmente é 100% compatível com a versão 7 dos recursos da linguagem, além de alguns recursos do Java 8 e 9. Em 2017, Kotlin foi adotada oficialmente como uma das linguagens oficiais do Android. Combinando os paradigmas de orientação a objetos (OOP) e programação funcional (FP), ela é mais concisa que a linguagem Java, além de evitar alguns recursos problemáticos como exceções de ponteiro nulo (NullPointerException). O Android Native Development Kit (NDK) é um conjunto de ferramentas que permitem usar código C e C++ em aplicativos Android. Devido à complexidade adicional inserida no processo de desenvolvimento ele agrega pouco valor para programadores novatos e para muitos tipos de aplicativos. Por outro lado ele pode ser essencial para jogos ou aplicativos intensivos em CPU, como simulações de física.

Na documentação do Android é possível encontrar guias com as diretrizes que compreendem um conjunto de práticas para auxiliar o desenvolvimento de aplicativos Android. Componentes arquiteturais (*Room, ViewModel, LiveData* e *Lifecycles*) foram criados para resolver cenários recorrentes de forma concisa,

menos propensos a bugs e problemas de desempenho. Um conjunto de critérios de qualidade como consistência do design com o SO, comportamento dos botões “Voltar” e “Home”, uso de notificações, permissões, desempenho e uso da bateria são descritos para auxiliar o desenvolvedor a incorporar os padrões de navegação e projeto do Android, além de ajudar na qualificação do aplicativo para oportunidades promocionais na Google Play Store.

As diretrizes de design são um complemento das diretrizes de desenvolvimento, trazendo documentação e recursos referentes ao material design e a experiência pura do Android. Essas recomendações são um esforço do Google em gerar uma experiência de usuário unificada e consistente em todo o Android. Entre elas estão guias para uso de componentes, estilo visual do aplicativo, animações, padrões de interação com o usuário, usabilidade, acessibilidade, layout e organização da informação.

2.2. iOS

Em 2005, quando o iPhone já estava sendo projetado, a Apple lançou uma competição interna para escolher o time responsável pelo desenvolvimento do SO. O time ganhador foi o mesmo time responsável pelo Macintosh, o que trouxe similaridades que permitiram com que desenvolvedores terceiros se sentissem familiarizados com a nova plataforma. Em 2007, juntamente com o lançamento do iPhone, o SO foi anunciado com o nome de iPhone OS. Inicialmente ele não tinha suporte a aplicativos nativos de terceiros e afirmava que o foco seriam aplicativos web no navegador Safari. Em 2008, após uma mudança estratégica, a Apple anunciou o iPhone Software Development Kit assegurando o novo foco em aplicações nativas. Neste mesmo ano também foi lançada a App Store inicialmente com 500 aplicativos. Em 2010, após negociar um licenciamento da marca com a Cisco, o iPhone OS foi renomeado para iOS.

O iOS recebeu novas versões anualmente desde seu lançamento trazendo melhorias tanto na interface com usuário quanto no sistema, das quais citaremos

apenas algumas. Inicialmente sem nome oficial, a primeira versão trouxe também a primeira versão mobile do Safari, navegador oficial da Apple. No ano seguinte foram introduzidos um SDK para aplicativos nativos assim como a App Store. Na versão 3, temos notificações push e suporte ao iPad na versão 3.2. A quarta versão marca a transição do nome para iOS, além de recursos importantes como multitarefa, o aplicativo FaceTime e o suporte a telas Retina. O iOS 5 apresentou a primeira versão da Siri, além de uma nova central de notificações. A sexta versão substituiu o Google Maps pelo aplicativo da casa Apple Maps, que gerou muita polêmica e culminou na troca de lideranças do projeto. O iOS 7 veio com um novo conceito de interface gráfica usando uma abordagem flat além do Touch ID. A versão 8 apresentou um sistema de comunicação chamado de Continuity que integrando Mac, iPhone e iPad permite que uma tarefa seja iniciada em um dispositivo e finalizada em outro. A nona versão trouxe o sistema de pagamentos Apple Pay. Os iOS 10 e 11 disponibilizam APIs para uso de aprendizado de máquina, realidade aumentada e da assistente virtual Siri.

O sucesso da App Store foi tão grande que inspirou o uso do termo “app store” por outros fabricante. A Apple até registrou a marca para garantir que só ela usasse o nome e chegou a processar a Amazon pelo uso indevido de sua marca registrada. Entretanto o processo não gerou resultados e foi abandonado dois anos depois. A App Store cobra uma taxa de $99 ao ano para que desenvolvedores possam publicar aplicativos. O faturamento é dividido na proporção 70/30 (desenvolvedores/Apple) e na proporção 85/15 para o modelo de assinatura. Em 2017 a loja alcançou a marca de 2,2 milhões de aplicativos e 130 bilhões de downloads desde seu lançamento tendo gerado $70 bilhões em receita para os desenvolvedores desde então.

O Software Development Kit do iOS já vem incluído na IDE XCode junto com o Simulator, um simulador de dispositivos Apple, como iPhone, iPad, Apple Watch e Apple TV. As linguagens de programação disponíveis para a programação de aplicativos iOS são Swift e Objective-C. O Software Development Kit do iOS somente está disponível para o sistema operacional macOS, entretanto é possível executar o macOS em uma máquina virtual, em um PC comum (Hackintosh) ou na

nuvem, onde é possível alugar um Mac remoto (hardware e software Apple) a partir de $20/mês. No caso de uma VM ou de um PC, vale lembrar que os termos de uso do macOS proíbem executá-lo em hardware não licenciado pela Apple.

O XCode é a IDE oficial para desenvolvimento de aplicativos Apple para os sistemas macOS, iOS, watchOS e tvOS. Lançado em 2003, atualmente está na versão 9 e suporta múltiplas linguagens além de Swift e Objective-C, como C++, Java, Python, Ruby, entre outras. Ela somente é compatível com macOS e pode ser baixada gratuitamente, desde que o desenvolvedor seja cadastrado (Apple ID).

Para a programação de aplicativos iOS temos duas opções de linguagens oficiais: Objective-C e Swift. Objective-C é a mais utilizada atualmente por ter sido a primeira linguagem suportada no iOS, além de ser usada no macOS desde o final dos anos 80. Por estar no mercado a mais tempo, ela oferece mais conteúdo para aprendizado (tutoriais, exemplo e blogs) além de mais bibliotecas de terceiros que complementam as funcionalidades do SDK. Lançada em 2014, a linguagem Swift apresenta uma sintaxe mais concisa, limpa e fácil de ler do que o Objective-C, além de realizar um tratamento moderno e seguro para problemas comuns em programação como ponteiros nulos. A linguagem também apresenta um gerenciamento automático de memória (ARC) em tempo de compilação que evita os vazamentos de memória do Objective-C e o overhead causado por garbage collectors (Java e cia).

Em 2016, a Apple apresentou o Swift Playgrounds, um aplicativo para aprendizado da linguagem de programação Swift e iOS, exclusivo para iPad. Otimizado para interação via toque, o aplicativo apresenta um ambiente interativo com lições passo a passo e desafios. Todos os exemplos e desafios utilizam código real em Swift e bibliotecas iOS. Seu conteúdo é organizado em níveis de dificuldade que vão do mais básico até conteúdos avançados. Ele também possui bibliotecas adicionais para o controle de robôs, drones e instrumentos musicais.

A Apple tem um controle de qualidade onde todo aplicativo submetido a App Store é cuidadosamente revisado. São analisados aspectos como conformidade com os padrões de design, segurança e privacidade, desempenho e estabilidade,

conteúdo (ofensivo, propriedade intelectual, etc.). Algumas das razões mais comuns para reprovação incluem aplicativo inacabado ou instável, spam, cópias de aplicativos existentes, descrições incompletas ou incorretas (metadados, screenshots), baixo valor agregado em funcionalidades e conteúdo ou um simples website sem cara de aplicativo.

Todos as plataformas Apple possuem um guia de interfaces humana (Human Interfaces Guidelines) que traz um conjunto de recomendações que visam melhorar a experiência do usuário através de interfaces intuitivas e consistentes. Essas diretrizes cobrem aspectos da arquitetura do aplicativo (como acessibilidade, feedback de carregamento de dados e janelas modais), interação com usuário (como 3D Touch, controle de áudio, autenticação e gestos), interação com o sistema (como multitarefa e notificações), visual (responsividade ao tamanho da tela, animações, cores e tipografia).

2.3. Web

A Web ou World Wide Web (WWW) é um sistema de documentos em hipermídia que são interligados e executados na Internet. Tais documentos podem ser hipertextos, imagens, áudio e vídeo. Criada por Tim Berners-Lee em 1989 no CERN, um centro de pesquisa na Suíça. O primeiro navegador web foi lançado em 1990 e divulgado publicamente apenas em 1991.

Ao longo dos anos passou por diversas evoluções que são divididas em duas grandes fases: Web 1.0 e Web 2.0. A Web 1.0, que representa o passado recente, era baseada em páginas com conteúdo estático somente para leitura sem interatividade do usuário com a página. Na Web 2.0, que representa o presente, as páginas possuem conteúdo dinâmico que pode ser alterado ou incrementado pelo usuário. Ela também é colaborativa, onde o usuário participa da construção do conteúdo. A Web 3.0, que representa o futuro, é semântica e tem a proposta de descentralizar serviços.

Navegadores (web) são programas para leitura e interação com páginas web. Os principais navegadores em 2017, segundo o site StatCounter, são: Chrome com 62,1% dos usuários, Firefox com 14,8%, Internet Explorer com 9,6%, Safari com 5,3%, Microsoft Edge com 3,7%, Opera com 1,6%, Outros com 2,9%.

Padronizar funcionalidades comuns a diversos navegadores é um desafio, que gerou abordagens de desenvolvimento como cross-browser e multi-browser. O World Wide Web Consortium, ou W3C, é uma organização internacional de padronização da Web fundada em 1994 que padroniza: HTML, CSS, SOAP, SVG, XML. A Ecma International é uma organização de padronização de sistemas informações fundada em 1961 e que padroniza JavaScript desde 1996, além de outras tecnologias como C#, Eiffel, FAT16, CD-ROM.

As principais tecnologias utilizadas na Web são HTML, CSS e JavaScript. O HTML ou HyperText Markup Language, é a linguagem usada para descrever e definir o conteúdo de uma página ou aplicativo. Ela é bloco básico de construção na Web, onde elementos são representados por tags (<head>, <title>, <body>, <header>, <footer>, <article>, <section>, <p>, , , <img>), as tags são organizadas hierarquicamente através de uma árvore de tags e tags possuem atributos que indicam informação extra aos elementos de uma página.

O CSS, ou Cascading Style Sheets, é a linguagem usada para descrever a aparência de um conteúdo. Ela especifica a renderização dos elementos na tela, no papel ou em outras mídias através de seletores e declarações. Seletores indicam o conjunto de elementos a aplicar uma ou mais declarações e declarações são pares de propriedade e valor que definem o estilo a aplicar nos elementos selecionados. O CSS é incluído no html através das tags <style> ou <link>.

O JavaScript é a linguagem usada para definir o comportamento de uma página ou aplicativo, permitindo a construção de interfaces interativas. Ela é baseada em protótipos para construção de objetivos e multiparadigma (OOP, imperativo, funcional). O código JavaScript de uma página pode ser incluído no html utilizando a tag <script>. A linguagem não está limitada aos navegadores Web,

podendo também ser utilizada em servidores (Node.js), jogos (Unity3D) e até mesmo sistemas embarcados (Tessel).

A Web não possui uma IDE oficial para o desenvolvimento de aplicativos sendo possível escolher entre uma vasta gama de opções. Uma delas é o Visual Studio Code, editor de código fonte gratuito e open-source com suporte a múltiplas linguagens, inclusive linguagens desktop. Ele possui recursos como code completion inteligente para HTML, CSS e JavaScript além de depuração JavaScript. Outras opções interessantes de IDE para o desenvolvimento Web são SublimeText e Atom.

Dispositivos possuem diferentes tamanhos de tela como desktops, tablets, TVs, wearables e são lançados novos tamanhos de tela todo ano. Páginas responsivas são um conceito desenvolvido para lidar com essa diversidade proporcionando fluidez e proporcionalidade ao invés de layouts com largura fixa. Para tal é necessário utilizar um recurso do CSS denominado media queries, onde regras de estilo são agrupadas segundo dimensões de largura e altura das telas dos dispositivos, combinados com larguras em percentual ao invés de valores absolutos.

Single page applications ou SPA são páginas ou aplicativos web cujo conteúdo é renderizado através de client side rendering (CSR), onde todo (ou quase todo) o código (HTML, CSS e JS) é obtido com um único carregamento de página e as transições de telas subsequentes não recarregam toda a página.

O oposto do CSR é o server-side rendering (SSR) que retorna uma página dinâmica já renderizada e pronta para visualização no navegador web. Também é possível realizar renderização parcial da página e combinar com CSR para mesclar o conteúdo em uma página ou layout previamente carregados. Essa técnica evita grandes esperas para primeira visualização de conteúdo, mas por outro lado aumenta o processamento realizado no servidor.

PWA, ou Progressive Web Application, é um termo cunhado em 2015 e indica aplicativos com as seguintes características: progressivo (funciona em todos os navegadores), responsivo (se adapta ao tamanho da tela do dispositivo),

independente de conectividade (funciona off-line ou em redes lentas), App-like (tem visual e comportamento semelhante a um app nativo), seguro (utiliza HTTPS), instalável (pode ser adicionados à tela inicial do dispositivo), linkavel (pode ser compartilhados por URL), entre outras.

2.4. Demais plataformas

Android Open Source Program (AOSP) é um programa que coordena o desenvolvimento open source do Android e permite versões customizadas do Android por fabricantes, como modificações na UI (skins), substituição de aplicativos padrão, troca da loja de aplicativos, remoção de aplicativos e drivers proprietários, sobrevida para dispositivos já sem suporte dos fabricantes e adaptações para mercados específicos como China e Índia. Como exemplos de plataformas derivadas do Android temos BlackBerry Secure da BlackBerry, ColorOS da OPPO, Google Pixel UI do Google, HTC Sense da HTC, LineageOS que é o sucessor do CyanogenMod (descontinuado), MIUI da Xiaomi, OxygenOS da OnePlus, entre outros.

O Tizen nasceu como um esforço conjunto entre Samsung e Intel e teve sua primeira versão em 2012. Sua proposta é suportar smartphones, tablets, smart TVs, multimídia veicular. É uma plataforma open-source baseada no kernel Linux (assim como o Android). Possui loja de aplicativos própria chamada Tizen Store. Suporta aplicativos nativos desenvolvidos com a linguagem C, aplicativos web desenvolvidos com HTML, CSS e JavaScript (e bibliotecas compatíveis) e híbridos com a linguagem C# (.NET) e o Xamarin.

O Windows 10 Mobile, chamado de Windows Phone inicialmente, possui código fonte fechado e proprietário e foi lançado originalmente em 2010. Seu foco inicial era em dispositivos Nokia, mas também saiu em dispositivos da HTC e da Samsung. Em 2015 foi renomeado para Windows 10 Mobile e em 2017 seu market-share caiu abaixo de 0,2%. Atualmente a plataformas está apenas em manutenção, sem desenvolvimento de novas funcionalidades.

O BlackBerry 10 é outra plataforma de código fonte fechado e proprietário. É baseado no SO QNX (adquirido em 2010) e utilizado apenas em smartphones e tablets fabricados pela BlackBerry. Seus aplicativos são desenvolvidos em C++ com framework Qt e possui compatibilidade com aplicativos Android (API 18). Foi muito relevante nos anos 2000, porém seu market-share encolheu desde 2016. A plataforma foi descontinuada e atualmente está apenas em manutenção, com a empresa focada no BlackBerry Secure (AOSP).

Firefox OS é uma plataforma open source anunciada em 2012 e que em 2015 já possuía 12 smartphones lançados em mais de 20 países. Seu foco era em aplicativos web, uma vez que a plataforma foi baseada no navegador web Firefox. O sistema operacional foi desenvolvido com base no kernel Linux combinado com algumas bibliotecas do Android como drivers para GPS, câmera, entre outros. Por falta de interesse do mercado a plataforma foi descontinuada em 2016. Seu sucessor, KaiOS, possui foco em dispositivos de baixo custo como os dispositivos com pouca memória e que não possuem touchscreen, denominados feature phones. Suas principais características são aplicativos web, porém com uma loja de aplicativos dedicada, suporte 4G, GPS e Wi-Fi, além de baixo consumo de bateria.

A plataforma MeeGo é o resultado da combinação dos sistemas operacionais Moblin, desenvolvido pela Intel, e Maemo, desenvolvido pela Nokia. Sua primeira versão foi lançada em 2010 e tinha o objetivo de ser o sistema operacional principal da Nokia substituindo o Symbian OS. Entretanto, seu desenvolvimento foi abandonado pela Nokia em 2011 e a última versão foi lançada em 2012. Sua proposta era suportar diversos dispositivos como smartphones, tablets, netbooks, TVs, multimídias veiculares, entre outros.

O sucesso da plataforma MeeGo é o sistema SailfishOS, que também é open source e baseado no kernel Linux. Ele foi criado por um grupo de ex-funcionários da Nokia que constituíram uma nova empresa, Jolla. O primeiro smartphone com o SailfishOS foi lançado em 2012, e o primeiro tablet lançado em 2015. A plataforma também é compatível com aplicativos Android e seus aplicativos nativos são

desenvolvidos em C++ com framework Qt. A plataforma foca em privacidade, personalização e multitarefa.

Ubuntu Touch é mais uma plataforma open source, anunciada em 2011 porém comercializada somente à partir de 2014. O sistema operacional pode ser usado com kernel Linux do Android o que permite a utilização de drivers e serviços Android porém sem aplicativos nativos em Java. A plataforma prometia suporte a smartphones, tablets, smart TVs e smartwatches, porém foi descontinuada em 2017 devido à falta de interesse do mercado.

Outra plataforma baseada no kernel Linux é o webOS, também conhecido por LG webOS ou Open webOS. Ele foi desenvolvido inicialmente pela Palm e comprado pela HP, quando foi utilizado em alguns smartphones e tablets em 2011. Foi licenciado para a LG em 2013 e atualmente é mais utilizado em smart TVs. Seu foco está em web apps com acesso ao hardware através de uma API própria da plataforma.

ChromeOS também é baseado no kernel Linux, com sua versão open source denominada Chromium OS. Foi anunciado em 2009, porém o primeiro dispositivo só foi lançado em 2011. Utiliza o Google Chrome como interface gráfica principal e suporta aplicativos web, Chrome Apps e aplicativos Android, a partir de 2016.

O Fuchsia é um sistema operacional de tempo real (RTOS) open source e desenvolvido pelo Google. Foi descoberto pela comunidade em 2016, sem anúncio oficial do Google. Seus aplicativos são desenvolvidos na linguagem Dart utilizando a SDK Flutter que também é multiplataforma. Ele não é baseado no Linux e sim em um novo microkernel denominado Zircon. A plataforma somente recebeu sua interface gráfica inicial em 2017 e ainda está em estágio inicial de desenvolvimento.

Capítulo 3. Estratégias arquiteturais e suas tecnologias

3.1. Aplicativos móveis nativos

Aplicativos móveis ou Mobile Apps são aplicativos instalados através de lojas de aplicativos como Google Play Store (Android), Amazon AppStore (Android), App Store (iOS) e Microsoft Store (Windows). Sua presença em tais lojas pode fortalecer a marca (branding) do produto. Aplicativos móveis utilizam notificações para manter o interesse do usuário pós instalação, onde estatísticas apontam um aumento de acesso em mais de 80% com o envio de notificações. Mobile Apps também permitem compartilhamento de conteúdo fácil e simples, já integrado à plataforma.

Aplicativos móveis nativos ou Native Mobile Apps são aplicativos móveis desenvolvidos nas linguagens nativas da plataforma e, portanto, interagem diretamente com os recursos da plataforma, incluindo hardware e demais aplicativos. Eles otimizam a experiência do usuário, pois são específicos por plataforma, utilizando IDE, SDK e interfaces de usuário desenvolvidas especialmente para a plataforma.

Para as principais plataformas do mercado (iOS e Android) as seguintes tecnologias são utilizadas: Swift e Objective-C no iOS através da IDE Xcode, e Java, Kotlin e C++ no Android através da IDE Android Studio.

Aplicativos Android em Java são compilados em um arquivo .apk e cada um é executado em um processo isolado com sua própria máquina virtual, seguindo o princípio do privilégio mínimo onde somente recursos essenciais para sua execução são disponibilizados. O arquivo AndroidManifest.xml identifica permissões do aplicativo, declara compatibilidade com versões Android, declara recursos de hardware e software utilizados como câmera e bluetooth) além de outras definições. Os componentes básicos de um aplicativo Android são: atividades (Activity) que correspondem às telas do aplicativo, serviços (Service) que definem processamento em segundo plano, provedores de conteúdo (ContentProvider) responsáveis pelo conjunto de dados compartilhado entre os componentes e receptores de

transmissão (BroadcastReceiver) que escutam eventos do sistema. Aplicativos podem iniciar atividades de outros aplicativos através do componente Intent, como por exemplo uma atividade de captura de fotos de um aplicativo de câmera ou uma atividade de seleção de um contato pessoal. O layout das telas em aplicativos Android é definido em arquivos XML.

Aplicativos Android em Kotlin seguem a mesma estrutura básica de um aplicativo Java. A linguagem Kotlin é interoperável com Java e pode ser usado em todo o aplicativo ou em apenas parte dele. O Android Studio pode converter arquivos Java em Kotlin facilitando a migração de projetos existentes. A linguagem é compatível com todas as versões do Android e causa um overhead no tamanho do APK devido ao ambiente de execução (Kotlin runtime) de 7K métodos e aproximadamente 1MB.

Aplicativos Android em C++ e NDK são normalmente desenvolvidos com o objetivo de portar aplicativos entre plataformas, reutilizar bibliotecas ou melhorar desempenho em caso de computação intensiva, como jogos. Eles apresentam comunicação Java/C++ através do JNI (Java Native Interface) e compilação específica para arquitetura (ARM, x86) segundo ABI configurado no projeto. Também é possível construir aplicativos sem código Java declarando a atividade NativeActivity no AndroidManifest.xml.

Aplicativos iOS em Objective C são compilados em um diretório que agrupa recursos do aplicativo, como executável, ícones, imagens, configurações, entre outros, denominado App bundle. O arquivo Information Property List File (Info.plist) é utilizado para declarar requisitos mínimos do app, ícones e splash screen, permissões e modo de segundo plano (Áudio, GPS, bluetooth, notificações remotas, etc.). A comunicação entre aplicativos pode ser feita utilizando AirDrop (arquivos) ou URL Schemes (http, mailto, tel, and sms). Os aplicativos seguem o padrão MVC, onde Model corresponde aos dados, View corresponde às interfaces e telas e Controller define como models e views devem ser atualizados. Storyboard são representações visuais da interface do usuário e podem ser definidas utilizando o editor visual no Xcode. Storyboards são compostos por Scene que definem telas

inteiras ou parciais (pares de View e Controller) e Segue gerenciam transições entre Scenes incluindo o tipo da transição (modal ou push) e transferência de dados entre telas.

Aplicativos iOS em Swift seguem a mesma estrutura básica de um aplicativo em Objective-C, incluindo o padrão MVC e a utilização dos Storyboards. A linguagem Swift é interoperável com Objective- C e pode ser usada em todo o aplicativo ou apenas em parte dele. Em alguns casos a linguagem Swift pode resultar em um desempenho 2.6x mais rápido que Objective-C.

Dentre as principais vantagens da construção de aplicativos nativos podemos citar o desempenho e responsividade devido à construção focada na plataforma alvo, a possibilidade de distribuição através de loja de aplicativo da plataforma, o fácil acesso ao hardware através das APIs nativas e a adoção de convenções de UI/UX da plataforma. Por outro lado é necessário construir uma versão por plataforma, o que implica em duplicar esforços para implementar novas funcionalidades e correções de bugs. Muitas vezes é necessário ter desenvolvedores especializados em cada plataforma, o que resulta em um maior custo de desenvolvimento e evolução. Pode ser necessário manter compatibilidade com novas e antigas versões da plataforma que ainda são muito usadas, pois nem todo dispositivo recebe as atualizações mais recentes das plataformas. A instalação do aplicativo ocupa armazenamento no dispositivo o que apresenta um problema para smartphones de baixo custo que normalmente possuem pouco espaço livre em disco.

3.2. Aplicativos móveis nativos multiplataforma

Aplicativos móveis nativos multiplataforma ou Cross-Platform Native Apps são aplicativos móveis nativos desenvolvidos para múltiplas plataformas usando uma linguagem comum, onde o código compilado para linguagem nativa da plataforma não utiliza o navegador web ou webview. Essa abordagem permite compartilhar mais de 90% do código fonte entre as plataformas.

Dentre as principais tecnologias utilizadas no desenvolvimento de aplicativos móveis multiplataforma podemos citar Xamarin da Microsoft com a linguagem C#, React Native do Facebook com a linguagem JavaScript e a biblioteca React, Titanium com a linguagem JavaScript, NativeScript também com JavaScript e Flutter da Google com a linguagem Dart.

Aplicativos Xamarin são desenvolvidos com a linguagem C# através da IDE Visual Studio e possui, além de ferramentas para build, testes e distribuição um ambiente de testes com milhares de dispositivos virtuais. Desde 2013 a plataforma oferece desenvolvimento de aplicativos Android e iOS. Foi comprada pela Microsoft em 2016 quando seu SDK foi disponibilizado como open source. A plataforma permite acesso a APIs nativas das plataformas, além de compilação otimizada por plataforma, o que resulta em um desempenho nativo. As telas da aplicação podem ser construídas programaticamente ou através de um editor visual com drag-and-drop que usa componentes nativos de cada plataforma gerando arquivos .XML (Android) e Storyboard (iOS). A estrutura do aplicativo é flexível e permite utilizar padrões arquiteturais como MVC, MVVM, entre outros. Também é disponibilizado um framework opcional para desenvolvimento de layouts compartilhados denominado Xamarin.Forms que aumenta ainda mais o reúso de código entre as plataformas.

React Native foi lançado em 2015 pelo Facebook e utiliza JavaScript e React sem fazer uso de HTML, DOM ou VirtualDom. O código JavaScript é interpretado no dispositivo ao invés e interage com componentes nativos de interface da plataforma. Essa abordagem permite interoperabilidade com código nativo e pode ser incluída em um aplicativo nativo existente. Aplicativos desenvolvidos com React Native são renderizados à partir de uma árvore de componentes construídos com React e JSX. É possível utilizar tanto componentes genéricos quanto específicos por plataforma. A transição entre as páginas pode utilizar diversas bibliotecas como react-navigation ou NavigatorIOS que é específica para iOS. Quanto ao gerenciamento do estado da aplicação ele é flexível sendo compatível com redux, muito utilizado em aplicativos web com React, e demais alternativas.

NativeScript foi lançado em 2015, é open source e utiliza a linguagem JavaScript que é interpretada no dispositivo. A biblioteca pode ser combinada com outros frameworks como Angular, na linguagem TypeScript, e Vue. Também pode ser integrada em um aplicativo nativo existente para estender suas funcionalidades. A interface com o a plataforma é realizada através de técnicas como reflection e metadados que garantem acesso imediato a novos recursos sem uma API intermediária para interface com os recursos que poderiam ficar desatualizados com as novidades das plataformas. Seus aplicativos são organizados em páginas cujo layout pode ser definido em arquivo XML ou usando código JS/TS. Uma tela pode ter diferentes arquivos XML diferenciados por prefixos que permitem customizações para diferentes tamanhos de tela (exemplo: main-page.minWH600.xml). A navegação entre páginas utiliza o objeto Frame que permite compartilhar informações entre páginas durante transição. A plataforma também permite utilizar os eventos de ciclo de vida específicos por plataforma como por exemplo activityPaused no Android e applicationDidBecomeActive no iOS.

O framework open source Flutter, desenvolvido pela Google, foi anunciado em 201, mas lançado apenas em 2017. Ele utiliza a linguagem Dart e é um framework reativo inspirado no React. Sua proposta é oferecer um desempenho próximo ao nativo utilizando compilação otimizada para as plataformas alvo com uma meta de renderização consistente da interface de usuário em 120 frames/s. Possui interoperabilidade com as linguagens Java, Objective-C e Swift além de APIs das plataformas nativas (iOS SDK, Android SDK). Seus aplicativos são construídos com widgets que são componentes que descrevem como a interface do usuário deve se parecer em uma dada configuração e estado. As mudanças de estado acionam atualizações nas que são propagadas para os widgets mantendo a UI consistente com o estado da aplicação. Widgets pode ser do tipo StatefulWidget, que armazenam o estado da aplicação, ou StatelessWidget que apenas renderizam informações na tela do dispositivo. A biblioteca possui tanto widgets genéricos como específicos da plataforma, como os widgets Material Components para o Android e os widgets Cupertino para o iOS.

Dentre as principais vantagens oferecidas por aplicativos multiplataforma temos a unificação das tecnologias utilizadas, o que permite reúso de código entre plataformas desde que a lógica da aplicação seja independente da plataforma. Essa característica resulta em um menor custo de desenvolvimento e manutenção. Por outro lado essa abordagem normalmente resulta em um desempenho e responsividade “quase nativos”, limitações de funcionalidades nativas em alguns casos, limitações de ferramentas de desenvolvimento, além de dependência de APIs de terceiros que podem ser descontinuadas. Em alguns casos algumas convenções de UI das plataformas podem trazer desafios de serem implementadas com fidelidade.

3.3. Aplicativos móveis híbridos

Aplicativos móveis híbridos ou Hybrid Apps são aplicativos web encapsulados em um aplicativo nativo, ou seja, são renderizados através de webview que é parte de um navegador web. Por estarem encapsulados em um aplicativo nativo permitem o download e instalação via loja de aplicativos da plataforma, e sua instalação no dispositivo consome armazenamento secundário. Em muitos casos são difíceis de distinguir de aplicativos nativos. A integração com hardware normalmente é realizada através de plugins que utilizam código nativo das plataformas. As principais tecnologias utilizadas atualmente são PhoneGap/Cordova, Framework7, Canvas e Ionic.

PhoneGap foi desenvolvido inicialmente entre 2009 e 2011 quando foi adquirido pela Adobe e então licenciado como open source com o nome Cordova. Atualmente o PhoneGap oferece serviços de build na nuvem (cloud compiler). A integração com a plataforma é realizada através de plugins que permite a utilização de acelerômetro, câmera, bússola, microfone, notificações, GPS, entre outros recursos. Essa abordagem permite uma construção flexível da aplicação sendo possível utilizar qualquer biblioteca ou framework JavaScript como por exemplo jQuery, Angular, React, Vue, entre outras. Cordova é também utilizado por outras abordagens híbridas como Ionic e Framework7.

Framework7 é um framework open source baseado no Cordova e compatível apenas com iOS e Android. Essa abordagem oferece rolagem de tela (scroll) nativa o que permite um desempenho muito próximo do nativo, além de componentes visuais e animações que parecem nativos no iOS. O framework também pode ser usado com as bibliotecas Vue ou React para construção das telas do aplicativo.

Canvas é uma solução proprietária desenvolvida pela empresa MobiLoud. Permite a interface com a plataforma através de APIs com suporte a notificações, modo off-line e cache. Essa abordagem necessita de um website remoto que é sincronizado com o aplicativo móvel. Os aplicativos são construídos sem necessidade de codificação através de uma conversão automática. O foco dessa tecnologia são aplicativos de notícias e blog que utilizam wordpress, comunidades que utilizam a tecnologia BudyPress e lojas virtuais construídas com WooCommerce.

Ionic é um framework open source e também baseado no Cordova que foi lançado em 2013 com integração com o framework JavaScript AngularJS e a partir da versão 2 utiliza o framework Angular com TypeScript. Ele também apresenta outras possibilidades de desenvolvimento através de subprojetos como o Stencil que construção de aplicativos independente do framework JavaScript e desse modo possibilitam uma alternativa ao Angular. Também é possível a construção de aplicativos nativos multiplataforma, uma alternativa ao Cordova, com a tecnologia capacitor. Aplicativos web progressivos também podem ser construídos com o PWA Toolkit, sendo uma alternativa aos aplicativos híbridos.

Aplicativos híbridos, apesar de serem aplicativos web internamente, apresentam algumas vantagens sobre aplicativos web como a não utilização de URL e interface de usuário do navegador e poderem ser disponibilizados através de lojas de aplicativos da plataforma. Quando comparados a aplicativos nativos temos um menor custo de construção e manutenção pois apenas uma versão é construída para todas plataformas. Também são de fácil adaptação para novas plataformas, além de ser possível realizar uma rápida conversão de um website ou aplicativo web para um aplicativo híbrido. Por outro lado, aplicativos híbridos possuem um maior

custo de desenvolvimento além da dependência de plugins e APIs de terceiros quando comparados a aplicativos web. E quando comparados a aplicativos nativos possuem desempenho inferior, acesso mais difícil ao hardware devido à baixa integração com as plataformas, além de diferenças de UI/UX podem incomodar e frustrar usuários leais.

3.4. Aplicativos web

Aplicativos web ou Web Apps são aplicativos acessados através de um navegador web pela Internet. Não necessitam de download e instalação anterior ao uso, assim como também não ocupam espaço no armazenamento secundário do dispositivo. Essa é uma definição ampla que pode servir tanto para aplicativos quanto para páginas web. Podemos diferenciá-los de acordo com o seguinte critério: páginas web focam na apresentação de um conteúdo, onde o usuário consome passivamente o mesmo, enquanto em aplicativos web o usuário interage ativamente com o conteúdo, seja na criação quanto na modificação e exclusão através de funcionalidades do aplicativo.

As principais tecnologias utilizadas para o desenvolvimento de aplicativos web são JavaScript, CSS e HTML. Existe uma ampla gama de bibliotecas e frameworks JavaScript para o desenvolvimento de aplicativos web como Angular, React e Vue. Também é possível traduzir outras linguagens diretamente para JavaScript como CoffeeScript, Dart, TypeScript, entre outras. No CSS a situação é semelhante com linguagens como Sass e Less. No caso do HTML temos linguagens para criação de templates como jade/pug, handlebars, marko, haml que podem ser compiladas para HTML.

Aplicativos web são compostos por um conjunto de arquivos javascript, css, imagens, html, fontes, entre outros. Quanto maior a complexidade, mais difícil gerenciar o desenvolvimento e deploy. O webpack é uma ferramenta de build para aplicativos web que monta um grafo de dependências entre recursos e otimiza carregamento de arquivos, utilizando técnicas como inlining e compressão. A

utilização de ferramentas de build em aplicativos web evita deploys com recursos ausentes ou desatualizados. O webpack também permite compilar o código fonte dos recursos, gerenciando transformações de SASS para CSS, JSX para JavaScript, entre outras. Existem outras ferramentas disponíveis no mercado que também realizam build de aplicativos web tais como rollup, browserify, gulp, grunt, parcel, entre outras.

React é uma biblioteca JavaScript para construção de interfaces de usuário open source lançada em 2013 pelo Facebook. Possui uma abordagem declarativa baseada em componentes e permite o desenvolvimento multiplataforma de aplicativos, podendo ser utilizada na renderização client-side (CSR), server-side (SSR) e inclusive aplicativos móveis (React Native). Aplicativos desenvolvidos com React possuem layout definido utilizando a linguagem JSX que é semelhante ao XML, porém é definido diretamente dentro do componente, ou seja, no meio do código JavaScript. Os dados seguem um fluxo unidirecional (one-way data flow) nos aplicativos, o que permite manter as interfaces de usuário sincronizadas com o estado da aplicação de modo eficiente utilizando uma técnica de verificação de atualizações conhecida como virtual DOM.

O Angular é um framework TypeScript para aplicativos web do tipo SPAs complexos que também é open source e foi lançado em 2016 pela Google, apesar de ter sido anunciado em 2014. Ele foi inspirado no seu antecessor AngularJS que apesar de ter nome semelhante é incompatível. Os aplicativos desenvolvidos com Angular são baseado em componentes e sua interface de usuário é definida em templates declarativos com sintaxe parecida com HTML. O framework utiliza técnicas como data binding para permitir a criação de fluxos unidirecionais e bidirecionais de dados entre componentes e templates. O Angular permite o desenvolvimento multiplataforma com suporte tanto a SSR e CSR quanto a aplicativos nativos com NativeScript e aplicativos híbridos com Ionic.

Outro framework JavaScript para construção de SPAs muito utilizado é o Vue, que também é open source e foi lançado em 2014. Ele também é baseado em componentes, além de ser progressivo e reativo, o que permite a adoção

incremental em projetos de aplicativos web. Interfaces de usuário são definidas através de templates declarativos usando HTML em conjunto com atributos html que indicam comportamento especial, conhecidos como diretivas.

Aplicativos web em geral permite um desenvolvimento simples e rápido, além de fácil manutenção e evolução, o que resulta em menor custo de desenvolvimento. Por não necessitarem de download e instalação prévia, além de serem naturalmente multiplataforma (navegador web), permitem um maior alcance em termos de usuários. Desse modo aplicativos web são uma ótima escolha para prototipação de ideias. Por outro lado seu uso exige navegadores e endereços URL, além de serem mais lentos que apps nativos, possuírem baixa integração com hardware com difícil acesso a câmera, microfone, acelerômetro, giroscópio, bússola, entre outros sensores. Aplicativos web também podem ter sua receita prejudicada por bloqueadores de anúncios (Ad Blockers) dependendo do modelo de monetização escolhido.

3.5. Progressive Web Application (PWA)

O maior desafio em projetos de software é a distribuição, uma vez que a disponibilidade em mais plataformas aumenta o alcance e portanto o potencial de receita do aplicativo. A web resolveu esse problema através da combinação navegador web e endereço URL que estão disponíveis em praticamente todos os dispositivo que possuem conexão com Internet. Web Apps chegam a ter 2,5 vezes mais visitantes únicos por mês que aplicativos nativos, porém os usuários passam dez vezes mais tempo em aplicativos nativos. Por outro lado cada plataforma apresenta um custo diferente de aquisição de usuários, sendo o custo de aquisição de novos usuários em aplicativos nativo dez vezes maior que aplicativos web. Um dos principais fatores que pode causar o abandono de um aplicativo web é o tempo de carregamento, onde estatísticas apontam que 40% dos usuários abandonam sites que demoram mais de três segundos para carregar.

Aplicativos web progressivos ou Progressive Web Applications (PWA) aproximam aplicativos web e nativos combinando o melhor de cada plataforma. Diferente de aplicativos web tradicionais, PWAs não exigem instalação e resultam em um desempenho melhorado, através de técnicas que reduzem o uso de dados, utilizam armazenamento do dispositivo através de cache inteligente, permitem funcionamento off-line resultando em um menor tempo de carregamento. Além disso, PWAs fazem uso de notificações push e permitem a inclusão do ícone do aplicativo na tela inicial do dispositivo.

O manifesto do aplicativo web é um arquivo JSON que permite controlar o seu comportamento em ambientes nativos, como a definição do modo e visual de inicialização, a personalização de ícones, nome e cores, a tela de apresentação e carregamento (splashscreen) e o modo da janela (standalone ou browser). Algumas dessas configurações são utilizadas para customizar o salvamento do aplicativo na tela inicial do dispositivo. Para ser utilizado o mesmo precisa ser referenciado no HTML do aplicativo, por exemplo através da tag <link rel="manifest" href="/manifest.json">.

Aplicativos PWA podem fazer uso de banners de instalação que permitem adicionar o aplicativo à tela inicial do dispositivo de maneira rápida e sem sair do navegador. A exibição dos banners é definida pelo navegador através de heurísticas como a presença do manifesto do aplicativo, a utilização de service worker assim como HTTPS, entre outras características. Existem dois tipos de banners, o primeiro permite adicionar o aplicativo na tela inicial do dispositivo através de um link de favorito, e o segundo permite instalar a versão nativa do aplicativo através de um link para a loja de aplicativos da plataforma.

App Shell Model se refere ao código HTML, CSS e JavaScript mínimos necessários para gerar a interface do usuário do aplicativo. Ele compõe o esqueleto mínimo da interface para que o aplicativo funcione e uma vez armazenado em cache permite o carregamento instantâneo do aplicativo. Normalmente é usado em conjunto com service workers, resultando em um desempenho confiável e consistente com interações próximas ao nativo. Também ajudam no uso econômico

de dados e podem ser utilizados também com renderização híbrida (CSR combinado SSR), onde no primeiro acesso é utilizado SSR e nos acessos seguintes, CSR.

Service Workers são um tipo de web worker, ou seja, scripts executados fora da thread principal do aplicativo e funcionam como um proxy client-side entre o aplicativo e o mundo externo como o server-side, demais APIs, entre outros. Permitem o controle granular sobre requisições de rede (http, sockets, etc.), além de cache diferenciado por tipo de recurso. Service Workers podem armazenar conteúdo para visualização off-line e também podem receber mensagens mesmo quando inativos (push messaging). Estão somente disponíveis em aplicativos servidos através de HTTPS e são instalados através do script principal do app apenas se o navegador tiver suporte. Service Workers somente são instalados se novos ou atualizados, e a instalação é feita em segundo plano.

Aplicativos web podem ser prejudicados por conexões instáveis ou inexistentes com a Internet assim como conexões com limite de dados. PWs permitem utilizar armazenamento off-line, podem aprimorar a experiência do usuário através de cache de recursos (html, css, js, imagens, etc.) e cache de conteúdo (json, xml, etc). Para tal podem ser utilizadas as APIs assíncronas compatíveis com service workers como Cache API que permite armazenar recursos endereçáveis por URL e IndexedDB que oferece um banco de dados que permite o armazenamento dos demais recursos. Cada navegador decide quanto espaço disponibiliza para os aplicativos utilizarem, como por exemplo no Chrome tem-se apenas menos de 6% do espaço livre, no Firefox menos de 10% do espaço livre e no Safari menos de 50MB.

Notificações push permitem reengajamento do usuário através de conteúdo customizado e relevante. A sua utilização é baseada em service workers, pois executam em segundo plano mesmo se o aplicativo estiver fechado. As principais APIs JavaScript necessárias para sua implementação são Push API, usada para receber conteúdo enviado pelo servidor, e a Notification API, usada para exibir o conteúdo ao usuário.

O acesso ao hardware dos dispositivos em aplicativos PWA é realizado através de APIs implementadas pelos navegadores web e padronizadas pelo W3C. Dentre as APIs com amplo suporte dos navegadores temos:

- Geolocation API que permite descobrir a localização do usuário e é suportada por todos os navegadores modernos (95% dos usuários),
- getUserMedia que permite usar câmera e microfone e é suportada por quase todos os navegadore exceto Safari <11 e IE (84% dos usuários),
- Orientação e movimentação do dispositivo que através do acesso ao acelerômetro, giroscópio e bússola permite detectar o lado que está para cima e como o dispositivo está girando, e possui suporte parcial na maioria dos navegadores (92% dos usuários)
- Vibration API que permite pulsar o dispositivo com duração variável e é suportada por quase todos os navegadores, exceto IE, Edge e Safari (76% dos usuários)

Algumas APIs ainda são experimentais e possuem suporte em apenas alguns navegadores além de não terem sido padronizadas como:

- Web Bluetooth que permite comunicação via Bluetooth com dispositivos próximos e é suportada apenas no Chrome
- WebUSB que permite comunicação com dispositivos via USB e é suportada apenas no Chrome
- Background Sync API que permite sincronização periódica em segundo plano usando Service Workers e é suportada apenas no Chrome
- Ambient Light API que permite acionar sensores de luz ambiente e é suportada apenas no Edge e parcialmente no Firefox
- Proximity API que permite acionar sensores de proximidade e é suportada apenas no Firefox

Resumindo, as principais vantagens dos aplicativos PWA quando comparados aos aplicativos nativos é que temos um menor custo de distribuição, conteúdo indexado facilmente por motores de busca (Google, Bing, etc.), qualquer página (URL) pode ser compartilhada facilmente e favoritada. Aplicativos PWA recebem atualizações automáticas a cada novo acesso, além de permitirem uma redução no tamanho do aplicativo e no consumo de dados, como no caso do PWA do Twitter que possui 600KB (a versão nativa possui 23,5MB) e permite uma economia de 70% com o PWA do Twitter. Entretanto PWAs não podem ser incluído nas lojas de aplicativos (Google Play, App Store, etc.) além de não possuírem acesso a recursos de hardware recentes como reconhecimento biométrico. Os navegadores web possuem ritmo diferente de desenvolvimento e por isso a compatibilidade de recursos pode estar limitada em alguns navegadores. Funcionalidades experimentais podem ser abandonadas ou alteradas o que gera um risco para adoção de novidades. Em alguns casos pode ser difícil alcançar bom desempenho em aplicativos complexos sem ajuda de especialistas.

3.6. Comparação entre as estratégias

A escolha da estratégia arquitetural depende dos objetivos de negócio, do público-alvo, da complexidade das funcionalidades, assim como do orçamento do projeto, entre outros fatores. Independentemente da estratégia, o resultado deve ser um aplicativo rápido, responsivo e confiável para que o usuário não abandone o seu uso. Cada estratégia oferece uma experiência diferente e é importante conhecer as vantagens e desvantagens de cada uma.

Dentre os principais critérios utilizados para a escolha da estratégia temos a experiência do usuário (UX), o desempenho e a responsividade do aplicativo, o uso do hardware, os custos de desenvolvimento, manutenção, contratação e treinamento de desenvolvedores, assim como o tempo necessário para levar o aplicativo ao mercado (Time-to-market).

Com relação à experiência do usuário, o melhor cenário é desenvolver aplicativos nativos especializados por plataforma, considerando que usuários se acostumam com o padrão da plataforma. Uma vez que plataformas possuem elementos visuais diferentes, será mais fácil adotar as convenções da plataforma com esta estratégia. As demais plataformas dependem de bibliotecas de terceiros para a padronização da interface, o que pode exigir personalização e desenvolvimento próprio, ocasionando o aumento no custo final do projeto.

Com relação a desempenho e responsividade, o melhor cenário também é desenvolver aplicativos nativos, principalmente para aplicativos que fazem uso de gestos para interação com usuário ou aplicativos intensivos em uso de CPU. Na prática, depende da complexidade do aplicativo, uma vez que a diferença pode ser imperceptível em aplicativos mais simples. É importante lembrar que os navegadores estão em constante evolução com novas técnicas de otimização sendo pesquisadas e aplicadas a cada nova versão, aproximando cada vez mais o desempenho do JavaScript das linguagens nativas.

No caso da utilização de recursos de hardware o melhor cenário é desenvolver aplicativos nativos ou multiplataforma, uma vez que ambos oferecem acesso completo a todos os recursos do hardware. No caso dos aplicativos híbridos o acesso depende de plugins de terceiros que podem sofrer com desenvolvimento estagnado além de problemas de qualidade e segurança. Aplicativos web dependem da implementação dos navegadores, o que pode gerar problemas de compatibilidade, mas que podem ser aliviados com técnicas progressivas (PWA) permitindo que o aplicativo se adapte aos recursos do navegador e use apenas o que estiver disponível sem interromper o funcionamento do aplicativo.

Os custos de desenvolvimento variam de acordo com a complexidade, funcionalidades e plataforma-alvo. Algumas estatísticas apontam que o desenvolvimento para Android chega a ser 30% mais caro que para iOS. Um dos motivos desse aumento de custos é resultado da linguagem nativa da plataforma, uma vez que Java é mais verboso que Objective-C e Swift. O Android também faz uso de emuladores que são mais lentos que os simuladores utilizados no iOS. Outro

motivo é a fragmentação da plataforma que exige compatibilização com versões antigas além da construção de layouts utilizando arquivos XML que não oferecem uma boa experiência de desenvolvimento. Cada plataforma exige um time especializado mesmo que o time seja apenas um desenvolvedor.

Quando nos referimos aos custos de manutenção, as plataformas nativas sofrem de fragmentação, ou seja, a base de usuários está espalhada em múltiplas versões consequência da falta de suporte às versões mais recentes em dispositivos antigos. Este problema, que é mais crítico no Android, torna mais caro manter múltiplas versões, que pode chegar a uma por versão do sistema operacional, e pode também causar impacto no suporte das APIs do backend da aplicação, além de dificultar a correção de bugs. Na prática, só afeta aplicativos que utilizam novos recursos da plataforma e cuja base de usuários esteja fragmentada, o que muitas vezes depende do público-alvo.

No caso dos custos de contratação e treinamento o melhor cenário é escolher abordagens que utilizam a mesma linguagem já dominada pelo time do projeto, seja ela aplicativo web, híbrido ou nativo multiplataforma, uma vez que aplicativos nativos exigem treinamento ou contratação nas linguagens específicas das plataforma-alvo, assim como desenvolvedores ou times especializados por plataforma e desenvolvedores tendem a se especializar em apenas uma das plataformas.

Se considerarmos o tempo necessário para levar o aplicativo ao mercado (time-to-market), o melhor cenário é escolher estratégias que permitem o desenvolvimento unificado, como aplicativo web, híbrido ou nativo multiplataforma. Aplicativos Web, assim como PWAs, possuem distribuição facilitada resultando em um maior alcance, além de não necessitarem aprovação de loja de aplicativos (Google Play, App Store) que podem impor restrições de conteúdo e necessitarem o pagamento de taxas. Aplicativos Web, por serem independentes da plataforma, não sofrem com os problemas de fragmentação de versões além de não exigirem instalação, permitindo alcançar usuários de dispositivos de baixo custo.

Capítulo 4. Estratégias para comunicação

4.1. BFF – Backend for Frontends

É comum começar um projeto web utilizando uma arquitetura monolítica, pois essa abordagem simplifica construção inicial de produtos ainda em validação. Por outro lado essa arquitetura acarreta em complexidade crescente e alto acoplamento à medida que o projeto cresce. Em alguns casos vale a pena refatorar o aplicativo utilizando uma arquitetura baseada em microsserviços que separa componentes em serviços bem-definidos, reduzindo o acoplamento do sistema e o impacto de alterações que agora ficam limitadas em contexto.

Alterações na interface de usuário (UI) são mais frequentes que no backend. Desse modo, desacoplar a UI da API aumenta produtividade pois simplifica alterações visuais e agiliza deploy. Como consequência do crescimento do uso de dispositivos móveis temos múltiplos clientes para uma mesma API, como aplicativos específicos por plataforma, além de aplicativos web e aplicativos móveis híbridos ou multiplataforma. Como resultado é possível que a API se torne muito genérica sendo necessário realizar muitas requisições para renderizar telas da UI. Neste cenário a API é o novo gargalo no desenvolvimento.

Como uma alternativa para resolver os problemas contextualizados acima foi proposta uma organização da arquitetura que, ao invés de utilizar uma API de propósito geral, utiliza um backend específico para cada cliente (aplicativo) que fica responsável por implementar a orquestração entre serviços, transformar dados para o formato necessário pelo cliente, agrupar dados e reduzir o número de requisições. Esse padrão ficou conhecido como Backend for Frontends (BFF) e foi descrito publicamente pela primeira vez em 2015 por Phil Calçado, na época desenvolvedor do SoundCloud, como um relato da experiência de refatoração de um aplicativo monolítico. O nome do padrão foi sugerido por Nick Fisher, na época líder da equipe web do projeto.

Na prática, quanto utilizado o padrão BFF, cada backend é parte da aplicação e é otimizado para a experiência do cliente. Os serviços utilizados por esses backends continuam desacoplados e independentes. O mapeamento entre cliente e backend não precisa ser necessariamente de um backend para cada cliente podendo também ser de um backend para um conjunto de clientes similares, como por exemplo um backend para aplicativos móveis (iOS e Android) ao invés de um backend para cada plataforma móvel.

O padrão BFF também pode ser utilizado como estratégia de migração e pode ser aplicado a arquiteturas monolíticas para ajudar na transição para microsserviços. Para tal é possível construir inicialmente um proxy que redireciona chamadas a API monolítica e novas funcionalidades serão desenvolvidas como serviços e aos poucos é possível realizar decomposições de funcionalidades existentes em novos serviços.

A implementação de múltiplos backends pode causar duplicação de código devido a codificação de agregações semelhantes de dados e interfaces com serviços em múltiplos cliente. Por outro lado, não usar o padrão BFF também pode causar duplicação de código, pois parte da lógica comum que acaba sendo implementada diretamente nos clientes. Além disso, pode ser mais difícil detectar tais duplicações devido ao uso de linguagens e tecnologias diferentes, como em aplicativos móveis nativos. Duplicação de código em arquiteturas baseadas em microsserviços ainda pode ser refatorada em um serviço comum aos backends caso o conceito possua mapeamento no domínio do problema, como por exemplo um serviço de lista de desejos que pode agregar serviços como inventário e catálogo de produtos.

A técnica Server Side Rendering (SSR) pode ser combinada com o padrão BFF possibilitando combinar agregação de dados e geração da UI em uma única requisição, o que reduz tempo total necessário para que o usuário visualize as telas do aplicativo, principalmente quando combinado com cache dos dados agregados. Entretanto essa abordagem pode aumentar tempo necessário para o envio do primeiro byte ao cliente de uma aplicação web.

A utilização do padrão BFF não está restrito às interfaces gráficas e também pode ser usado para isolar chamadas de serviços de terceiros, o que reduz acoplamento, restringe o acesso aos dados (API), pode resultar em mais desempenho e segurança além de reduzir problemas de compatibilidade com versões legadas.

É comum serviços e UI evoluírem em ritmos diferentes e neste caso o padrão BFF permite configurar times de desenvolvimento mais independentes evitando que um time interrompa o desenvolvimento para esperar outro e permitindo que o time de um cliente mova funcionalidades do cliente para o backend com autonomia.

O padrão BFF deve ser utilizado com cuidado, pois em aplicativos com somente um cliente camada extra pode aumentar latência e trazer pouco valor. Entretanto, caso o cliente necessite de muita agregação de dados no server-side, a aplicação do padrão ainda pode trazer benefícios. Quando existem múltiplos clientes, tanto UI quanto serviços de terceiros, a separação de interesses resultante é benéfica na maioria dos casos, exceto quando o custo de deploy for muito alto.

4.2. Serverless

As arquiteturas serverless não removem os eliminam a necessidade de servidores para o backend de um aplicativo, apesar do termo serverless de traduzido como “sem servidor”. A diferença é que agora o server-side da aplicação é gerenciado por serviços ou aplicativos de terceiros (serverless databases) ou ainda quando parte da lógica server-side desenvolvido pelo time, mas executado por serviços de terceiros (serverless runtime). Serverless indica uma solução de computação em nuvem com gerenciamento dinâmico da alocação da infraestrutura movendo o foco dos esforços do time no que oferece mais valor aos seus usuários através da terceirização de serviços comuns como bancos de dados, índices de pesquisa, filas, mensagens SMS e entrega de e-mail.

Existem dois tipos de arquiteturas serverless: serverless runtime e serverless database. O primeiro também é conhecido pelo termo FaaS ou Function as a

Service e permite terceirizar a execução da lógica de aplicação sem armazenamento de dados. O serviço Zimki, lançado em 2006, foi a primeira plataforma de execução “pay as you go” baseado em JavaScript mas que não foi comercialmente bem sucedida. Em 2008 temos o lançamento do Google App Engine que era baseado em um framework Python personalizado que não permitia código arbitrário e apresentava cobrança por uso. Posteriormente foi adicionado suporte à mais linguagens e frameworks. Em 2014 o serviço AWS Lambda foi introduzido pela Amazon como o primeiro a oferecer abstrações de execução (funções). Inicialmente com suporte apenas para JavaScript/Node.js e atualmente com suporte a Python, Java, C# e Go, além de permitir que demais linguagens sejam suportadas indiretamente através de Node.js. OpenWhisk é uma plataforma open source publicada pela IBM com suporte nativo para JavaScript/Node.js, Python, Java e Swift, e suporte às demais linguagens e ambientes por meio de containers Docker. Outro exemplo de serviço serverless é o Azure Functions da Microsoft que suporta as linguagens C#, F#, Node.js, Java e PHP.

Serviços serverless runtime são frequentemente comparados a PaaS (Platform as a Service) e a containers. Quando comparados com PaaS temos pouca diferença na implementação do aplicativo, pois ambos recomendam a utilização da abordagem 12-Factor App, onde os aplicativos devem ser modelados como processos stateless permitindo fácil escalabilidade horizontal. A maior diferença acontece na operação onde PaaS exige pensar na escala (exemplo: quantos ativos Dynos no Heroku) e FaaS possui um controle granular nível de requisição que permite escalabilidade transparente além de ser mais eficiente em custos. Semelhante a comparação com PaaS, containers ainda não oferecem escalabilidade granular, transparente e automática. Entretanto essa diferença no gerenciamento pode se estreitar no futuro, como o desenvolvimento do “Horizontal Pod Autoscaling” no Kubernetes. A escolha pode estar mais ligada ao tipo da aplicação, onde FaaS poderia ser usado para aplicativos com poucos eventos por componente e containers para aplicativos com muitas requisições síncronas, além de ser possível usar as duas abordagens em um mesmo aplicativo.

Serverless databases são também conhecidos pelo termo Backend as a Service (BaaS) e eliminam necessidade de fornecer ou dimensionar infraestrutura de banco de dados. Nesta abordagem o banco de dados é ligado, desligado e escalado automaticamente de acordo com o uso o que a torna ideal para cargas de trabalho não frequentes, intermitentes ou imprevisíveis. Entre os muitos serviços desse tipo no mercado temos o Google Cloud Datastore, que é o componente de banco de dados do Google AppEngine disponibilizado como um serviço independente. FaunaDB apresenta um banco de dados NoSQL transacional distribuído globalmente e desenvolvido pelo time que escalou o Twitter. Amazon Aurora Serverless é banco de dados relacional compatível com MySQL e PostgreSQL. Azure Data Lake é serviço de armazenamento e análise de dados estruturados, semiestruturados e não estruturados.

Alguns dos principais benefícios da utilização de arquiteturas serverless incluem a redução do custo operacional (tanto em custos de infraestrutura quanto em recursos humanos através da terceirização que permite efeitos de economia de escala), a escalabilidade horizontal automática, elástica e transparente que é gerenciada pelos provedores de serviço, o pagamento pelo uso que permite a cobrança pelo tempo de processamento da função (dependendo do tipo de tráfego pode representar uma grande economia), a redução do custo e tempo de desenvolvimento (funcionalidades comuns podem ser terceirizadas, como autenticação, log e analytics resultando na redução no tempo entre ideia e produto), a redução do custo de deploy (é mais simples publicar uma função do que um servidor inteiro), além da possibilidade de usar várias linguagens de programação principalmente em times autogerenciáveis em que o próprio time escolhe a linguagem utilizada (Java, JavaScript/Node.Js, Python, C#, etc.).

Por outro lado, abordagens serverless aumentam o controle do provedor, deixando o aplicativo sujeito a limitações arbitrárias, alterações nos preços e alterações nas funcionalidades oferecidas. As tecnologias proprietárias podem ser incompatíveis e impedir ou dificultam a migração entre provedores. O monitoramento e depuração pode estar limitado às ferramentas disponibilizadas pelo provedor, uma vez que o ambiente de execução pode ser proprietário, tornando

o diagnóstico de desempenho ou problemas de uso excessivo de recursos mais difícil do que servidores tradicionais. Em aplicativos multi tenancy (diferentes clientes compartilhados), a segurança pode ser comprometida com clientes acessando dados de outros clientes, a robustez pode ser prejudicada quando um erro causado por um cliente pode afetar os demais e o desempenho pode ser afetado quando um cliente consumindo mais recursos pode prejudicar os demais. Ainda sobre o desempenho, código serverless usado com pouca frequência possui latência de resposta maior que o código que está sendo executado continuamente em um servidor dedicado. Quando falamos sobre deploy muitas vezes temos o deploy individual por função (Faas) apesar de existirem alternativas open source e proprietárias para minimizar este problema.

Voltando na questão da segurança, temos que o provedor da nuvem é responsável pelo monitoramento e atualização do SO e dependências. Em abordagens serverless, o aplicativo é composto por muitos outros componentes em comparação com as arquiteturas tradicionais, o que resulta em uma maior superfície de ataque, sendo que cada componente é um ponto de entrada para o aplicativo. Soluções de segurança que os clientes costumavam ter para proteger suas cargas de trabalho da nuvem tornam-se irrelevantes e clientes não conseguem controlar e instalar qualquer coisa no ponto final e no nível da rede, como sistemas de detecção e prevenção de intrusos (IDS / IPS).

Para tornar a criação, teste e implantação de aplicações serverless mais fáceis foram desenvolvidos frameworks serverless como uma camada de abstração independente do provedor de serviços. Com o desafio de padronização entre interfaces proprietárias, frameworks serverless evitam acoplamento com um único provedor (vendor lock-in) através da abstração de detalhes da implementação permitindo a implantação do aplicativo em um único pacote em qualquer provedor de nuvem.

Como exemplo de frameworks Serverless temos o Serverless Framework que foi pioneiro e inicialmente focado no AWS Lambda embora atualmente suporte Azure Functions, OpenWhisk, Google Cloud Functions, entre outros. Kubeless

trouxe uma prova de conceito para desenvolver um serverless framework para Kubernetes. Up apresenta uma ferramenta de código aberto para implementação de servidores HTTP para AWS Lambda. Nuclio é um framework de código aberto que executa até 400.000 invocações por segundo. OpenFaas traz um framework de código aberto para construir funções serverless com Docker e Kubernetes e que possui suporte de primeira classe para métricas. E por último temos o Fission, um framework para funções serverless no Kubernetes, suportando Python, NodeJS, Go, C# e PHP.

4.3. Realtime databases

Bancos de dados de tempo real ou Real-time databases possuem processamento em tempo real para cargas de trabalho cujo estado muda constantemente, ou seja, os registros são alterados ou incluídos constantemente e as transações são processadas com tempo de resposta previsível. As transações devem respeitar as propriedades ACID: Atomicidade (transações são indivisíveis), Consistência (execução de uma transação mantém consistência), Isolamento (transações paralelas não interferem umas nas outras) e Durabilidade (transações causam efeitos permanentes).

Tais bancos de dados podem ser categorizados segundo suas restrição de tempo e prazos. O tipo Hard indica que as restrições de tempo são rigorosas com tempos de resposta garantidos, sendo utilizados em sistemas de tempo real críticos ou rígidos, como sistemas de aviso e alarmes de emergência, máquinas industriais, robôs. Os de tipo Firm abortam transações não completadas antes do prazo. No caso do Soft, as transações perdidas degradam apenas a qualidade do sistema, e são utilizados em sistemas de tempo real não críticos ou moderados como sistemas bancários, sistemas de reservas aéreas.

Entre as principais características de bancos de dados de tempo real temos que eles devem manter a consistência lógica e temporal dos dados, manter as propriedades de temporização de transações, realizar consultas que respeitem

prazos definidos (soft ou hard), retornar dados que podem ter consistência absoluta e relativa, tratar dados temporais que se tornem desatualizados, pois nem todos os dados são permanentes e alguns podem ser temporais, como por exemplo dados de sensores, cotação de ações entre outros.

Bancos de dados de tempo real facilitam a criação de aplicações reativas onde alterações realizadas nos dados são imediatamente propagadas para os clientes (client-side) permitindo um comportamento ativo onde o banco de dados avisa cliente sobre mudanças (push-oriented). Em bancos de dados tradicionais o comportamento é passivo onde o cliente pede as informações que necessita (pull-based) o que exige um controle externo para manter os cliente atualizados, como por exemplo middlewares no backend combinados com websockets.

Consultas em tempo real são baseadas em fluxos de eventos (event stream), onde inicialmente enviam os dados mais atuais e posteriormente enviam novos dados caso haja novos registros, alterações em registros existentes ou registros removidos (não suportado em todos as tecnologias). Cada cliente (aplicativo) é notificado a cada alteração e fica responsável pela atualização da UI.

Dentre as tecnologias mais utilizadas para implementação de aplicativos reativos com bancos de dados de tempo real temos RethinkDB, Firebase, Baqend, CouchDB e MongoDB (3.6+). No RethinkDB temos um banco de dados orientado a documentos com documentos JSON com esquema dinâmico. Esse banco de dados foi projetado para facilitar a sincronização de atualizações com clientes e é recomendado para aplicações como aplicações web ou móveis reativas e colaborativas, jogos multiplayer, marketplaces de tempo real (exemplo compra e venda de ações), painéis de controle em tempo real (indicadores de negócios) e dispositivos conectados (IoT). Essa tecnologia não é recomendada para aplicações que necessitem de garantias fortes nos esquemas de dados como oferecidos por bancos de dados relacionais, análise de dados computacionalmente intensiva como oferecido por sistemas como Hadoop e alta disponibilidade para escritas como o banco de dados distribuído Riak.

Firebase disponibiliza um banco de dados NoSQL em tempo real como serviço (BaaS) onde dados são sincronizados em tempo real com todos os clientes conectados. Nele é possível escolher entre dois modelos de dados: Realtime Database e Cloud Firestore. No primeiro os dados são armazenados como uma grande árvore JSON permitindo um armazenamento fácil de dados simples porém inadequado para dados complexos e hierárquicos. No segundo temos um armazenamento de documentos organizado em coleções, onde dados simples armazenados em documentos usando JSON e dados complexos e hierárquicos são organizados em sub coleções. A integração com aplicativos pode ser feita através de API REST assim como bibliotecas para integração linguagens e plataformas (Java, Android, Objective-C, Swift, iOS, JavaScript e Node.js). O Firebase permite retenção de dados off-line com cache local (bibliotecas), o uso de filtros e ordenação de dados assim como uso direto através do frontend (client-side). O serviço ainda inclui mecanismos de autenticação e autorização o que facilita o desenvolvimento de aplicativos móveis ou web sem a necessidade de um backend dedicado.

Outro exemplo de banco de dados de tempo real é o Baqend, um serviço de banco de dados com consultas em tempo real (BaaS) que utiliza particionamento de objetos para garantir escalabilidade e alto desempenho respeitando latência e taxa de transferência necessários. Ele utiliza a arquitetura InvalidDB que distribui objetos e consultas em uma estrutura bidimensional onde número de partições é configurável e mais partições resultam em mais escalabilidade.

Duas tecnologias que não foram construídas como bancos de dados de tempo real porém apresentam algumas funcionalidades desse estilo são o CouchDB e o MongoDB. O CouchDB apresenta um banco de dados com arquitetura NoSQL orientada a documentos onde JSON é utilizado para armazenamento de dados e JavaScript como linguagem de consultas. Possui tanto uma API para envio de notificações de alterações nos documentos, onde a consulta inicial deve ser feita separadamente e não notifica sobre remoções de documentos. O MongoDB traz um banco de dados com arquitetura NoSQL orientada a documentos onde os dados são armazenados em documentos semelhantes a JSON. A partir da versão 3.6 ele disponibiliza uma API para obter um fluxo de eventos com alterações nos

documentos que é filtrável através de uma consulta e retorna os campos que mudaram nas transação de atualização. Os eventos propagados podem ser de cinco tipos diferentes: inserção, atualização, substituição (atualização que substitui o documento inteiro), remoção ou invalidação (remoção do banco de dados ou da coleção de documentos).

Capítulo 5. Estratégias para armazenamento local

5.1. SQLite

O SQLite é banco de dados relacional que não utiliza arquitetura cliente-servidor e implementa a maior parte do padrão SQL. Possui transações ACID (atômicas, consistentes, isoladas e duráveis) mesmo após falha do aplicativo, sistema operacional ou falta de energia. É muito usado para armazenamento local em navegadores web, sistemas operacionais, sistemas embarcados, dispositivos móveis, entre outros. Diferente de bancos de dados com arquitetura cliente-servidor, o SQLite é incluído diretamente no programa final e é acessível através de chamadas de função da sua biblioteca o que resulta em baixa latência nas consultas. O banco de dados é armazenado por inteiro em um único arquivo, incluindo definições, tabelas, índices e dados, e também é bloqueado por inteiro durante escritas, ou seja, apesar de leituras poderem ser realizadas em paralelo, as escritas somente são realizadas sequencialmente.

Entre suas características temos um biblioteca compacta (menos de 1 MB) na linguagem C, suporta de até 140 TB de dados, tamanho máximo das linhas (rows) de 1GB, arquivo multiplataforma que pode ser copiado livremente entre sistemas 32-bit e 64-bit ou arquiteturas big-endian e little-endian. O SQLite pode ser 35% mais rápido do que escrever diretamente no disco além de poder reduzir espaço em disco em 20% para arquivos entre 250KB e 1MB.

Sua configuração é mais simples do que bancos de dados do estilo cliente-servidor sendo chamada de Zero-conf pois não necessita de instalação anterior ao uso, procedimentos e arquivos de configuração, gerenciamento de serviços (iniciar ou pausar processos), procedimentos de recuperação após falhas de sistema ou energia, além de controle de acesso GRANT e senhas (gerenciado pelas permissões do sistema de arquivos).

Os principais cenários de uso para escolha do SQLite são sistemas embarcados e IoT como smartphones, TVs, videogames, câmeras, relógios,

termostatos, automóveis, sensores, drones, robôs. Seu uso não é recomendado quando a aplicação e os dados são separados por uma rede, sendo neste caso melhor usar banco de dados com arquitetura cliente-servidor. Também não é o mais adequado quando a aplicação for intensiva em escritas, pois suas escritas são sequenciais. O melhor cenário para o SQLite envolve consultas simples com baixa concorrência geram benefícios de desempenho por evitarem comunicação inter processos.

O SQLite utiliza PostgreSQL como plataforma de referência para sua implementação do padrão SQL, entretanto existem algumas diferenças como por exemplo na checagem de tipos onde no SQLite os valores são dinâmicos e conversões automáticas são realizadas durante o armazenamento de dados em cenários potencialmente reversíveis. Em geral é aplicada a Regra de Postel ou Princípio da robustez onde o banco é conservador nas operações realizadas (envia dados seguindo os padrões) e é liberal nos dados que aceita (recebe dados fora dos padrões). Sendo assim o SQLite implementa a maior parte do padrão SQL-92 exceto triggers, escritas em views, além de não permitir modificar ou excluir colunas de tabelas (ALTER TABLE) e possuir tipagem dinâmica de valores individuais ao invés de tipos em colunas nas tabelas onde é possível, por exemplo, inserir uma string em uma coluna de inteiros. A tipagem dinâmica de valores adiciona flexibilidade principalmente quando combinado com linguagens com tipagem dinâmicas como JavaScript e Python. Entretanto esta técnica reduz portabilidade do banco de dados. O SQLite também suporta conteúdo JSON a partir da versão 3.9.

Entre as possíveis integrações com outras linguagens tempos APIs C/C++, Tcl, C#/.NET, entre muitas outras linguagens, além de suporte em plataformas móveis nativas (Android e iOS) e suporte na web através do Web SQL. No Android, o cenário de uso mais comum do SQLite é o cache de dados para permitir funcionamento off-line. Disponível no pacote android.database.sqlite através da classe SQLiteOpenHelper que oferece APIs para abertura e fechamento do arquivo do banco de dados e execução de comandos SQL entre outras funções. O banco de dados é armazenado na pasta protegida do aplicativo que não é acessível a outros aplicativos ou ao usuário. O Android SDK também inclui um utilitário (sqlite3) para

listar conteúdo das tabelas e executar comandos SQL facilitando o desenvolvimento do aplicativo. O Android também oferece uma camada de abstração sobre o SQLite baseada em anotações através do componente Room. Essa camada é dividida em 3 macro componentes: Database que serve como ponto principal de conexão com o banco de dados e lista as entidades (Entity) disponíveis, Entity que representa uma tabela e suas colunas no banco de dados e DAO que contém métodos de acesso ao banco de dados e consultas SQL.

No iOS o acesso é direto à api C do SQLite e não possui abstrações nativas específicas para o SQLite apesar de existirem bibliotecas de terceiros para abstração da manipulação do SQLite como FMDB. O iOS também possui uma alternativa de armazenamento nativa chamada de CoreData que apresenta framework para armazenamento de um grafo de objetos e pode ser serializado em XML, arquivo binário ou SQLite. Apesar de usar o SQLite, o CoreData não é considerado um banco de dados.

O SQLite também pode ser utilizado em apps híbridos e multiplataforma como Xamarin através das bibliotecas ADO.NET, SQLiteNET e SQLite.NET PCL (Xamarin.Forms), no Ionic e demais frameworks baseados no Cordova através de plugin de terceiros (cordova-sqlite-storage), no NativeScript e no React Native também através de plugins de terceiros (nativescript-sqlite e react-native-sqlite-storage).

5.2. Alternativas SQLite em aplicativos móveis

O SQLite é o banco de dados mais utilizado por aplicativos móveis e sua utilização traz vantagens como a inclusão nativa no Android e iOS, a linguagem SQL que é expressiva e bem estabelecida, controle completo das consultas SQL e fácil de depuração além de ser uma tecnologia estável e bem testada. Por outro lado possui uma utilização verbosa (boilerplate), não possui verificações em tempo de compilação das consultas SQL, alguns desenvolvedores não dominam SQL e consultas SQL podem ficar longas e complicadas. Com a proposta de resolver

algumas dessas desvantagens além de trazer novos recursos surgiram alternativas como Couchbase Lite, LevelDB, Realm, ObjectBox entre outras.

Couchbase Lite é um banco de dados de documentos JSON que utiliza uma representação de dados com esquema flexível, ou seja, não exige um layout de dado pré-fixado. Pode ser usado tanto em modo standalone quanto em redes P2P ou como endpoint remoto para um gateway de sincronização. Disponibiliza APIs nativas para diversas linguagens e plataformas como iOS/tvOS/macOS, Java e Android, .NET e Xamarin, além de JavaScript, Cordova/PhoneGap e Ionic que permite manipulação direta de estruturas JSON, mapeamento de documentos para o modelo de objetos nativo da linguagem ou ambas combinadas, além de oferecer uma API REST para desenvolvimento de aplicativos híbridos. Suporta replicação com servidores de banco de dados compatíveis que permite a sincronização de dados entre múltiplos dispositivos e usuários. Também suporta acesso offline aos dados, permitindo funcionamento baseado em dados locais resultando em responsividade independente da conectividade de rede, onde é possível modificar dos dados enquanto offline e as alterações são sincronizadas com servidor quando possível.

O Couchbase Lite permite criptografar o banco de dados inteiro usando o algoritmo AES-256 com pequeno overhead de 5 a 15% no desempenho do banco. A criptografia é recomendada quando houver armazenamento de dados privados ou confidenciais ou regulamentações que exigem criptografia de dados e não é recomendado quando houver armazenamento de dados públicos ou não confidenciais, quando a criptografia do dispositivo for suficiente para os casos de uso (exemplo iOS) quando o gerenciamento de senhas incomodar o usuário (solicitado a inserir senha para acessar dados) ou mesmo quando a perda de desempenho causada pelo overhead impactar na aplicação.

O LevelDB é uma biblioteca de persistência de dados leve e compatível com múltiplas plataformas que armazena mapa de chaves e valores arbitrários e permite conteúdo de qualquer tipo, desde strings ASCII até arquivos binários. Os dados são armazenados com ordenação lexicográfica da chave por padrão e é possível

customizar as funções de comparação para a ordenação. Permite múltiplas mudanças em uma única operação atômica além de compressão opcional automática de dados através da biblioteca Google Snappy que oferece um algoritmo de compressão otimizada para desempenho resultando em baixo overhead no desempenho do banco de dados. O LevelDB não é um banco de dados SQL e portanto não possui um modelo relacionado de dados, não permite consultas SQL e não suporta índices. Cada banco de dados LevelDB somente pode ser utilizado por um único processo que apesar disso pode ser multi-thread. Não possui suporte nativo à arquitetura cliente-servidor e o aplicativo precisa oferecer a própria camada de servidor envolvendo a biblioteca, o que permite ser utilizado em rede usando protocolos como http, tcp ou udp.

Realm é um banco de dados open source para armazenamento de dados através de objetos. Inicialmente focado em dispositivos móveis, atualmente é compatível inclusive com aplicativos desktop. É muito utilizado para aplicações móveis com abordagem offline-first. Possui integrações nativas com Java, Swift e Objective-C, JavaScript e Xamarin. Permite relacionamento entre objetos através de links que criam um relacionamento bidirecional automático. A definição do esquema de dados é feita de acordo com a linguagem utilizada, onde herança de classes é utilizada em linguagens com tipagem estática (Java, Swift, etc.) e objetos em linguagens com tipagem dinâmica (JavaScript). Realm também permite criptografia opcional e configurável por esquema, além de facilitar a criação de aplicativos reativos onde objetos estão sempre atualizados e notificações de alterações permitem atualizar a UI do aplicativo. Realm também permite acompanhar progresso da sincronização de dados, ou seja, o status dos downloads e uploads de dados entre cliente e servidor, além de também permitir a sincronização parcial de dados, sendo possível inclusive separar e não sincronizar dados que são somente locais.

ObjectBox é um banco de dados open source orientado a objetos que não utiliza SQL, não necessita de camadas de abstração e transformação (ORM), suporta relacionamentos entre objetos e foi construído a partir de experiências com desenvolvimento de um ORM para o SQLite e Android. Assim como no SQLite, o

ObjectBox armazena todo o banco de dados em um único arquivo. Mas diferente do SQLite, o ObjectBox permite verificar consultas em tempo de compilação utilizando o componente QueryBuilder além de não necessitar de migrações manuais de esquemas, pois realiza versionamento automático de objetos e permite que objetos possam ter atributos incluídos, renomeados e removidos. O ObjectBox possui suporte a diferentes linguagens e plataformas como Android (Java e Kotlin), Linux (Java), Windows (Java) e macOS (Java).

5.3. Web Storage, Web SQL e IndexedDB

Navegadores web implementam tecnologias que permitem aos aplicativos web realizar armazenamento local de dados. Entre elas temos Web Storage, que inclui localStorage e sessionStorage, além de Web SQL e IndexedDB.

O WebStorage é utilizado para armazenar dados em área separada por origem ou domínio. Os dados não são enviados nas requisições ao servidor e o servidor não pode escrever diretamente no armazenamento, diferente dos cookies. No sessionStorage os dados são temporários e mantidos enquanto o navegador estiver aberto, já no localStorage os dados são persistidos mesmo após fechamento do navegador. O localStorage representa o dados através de modelo de dados de arranjo associativo com chaves e valores que devem ser strings. Ele é compatível com todos os navegadores modernos alcançando 93% dos usuários e é útil para armazenamento de pequenas quantidades de dados não estruturados.

O Web SQL apresenta um banco de dados relacional com implementação nos navegadores baseada no SQLite, onde a linguagem de consulta SQL suportada pelo SQLite 3.6.19, e possui suporte a tabelas, colunas e transações, além de possuir uma API assíncrona que permite a utilização em service workers. É possível criar múltiplos bancos de dados que somente são acessíveis criados na mesma origem onde foram criados. A sua padronização pelo W3C foi interrompida pois a especificação chegou em um impasse devido à falta de implementações independentes pois todas foram baseadas no SQLite. O Web SQL não é suportado

por todos os navegadores e está disponível apenas no Chrome (incluindo Android), Opera e Safari (incluindo iOS) alcançando cerca de 77% dos usuários web.

O IndexedDB é um banco de dados transacional orientado a objetos que não utiliza SQL, armazena e recupera objetos indexados por uma chave e permite que valores sejam objetos com estruturas complexas, além de permitir que chaves sejam objetos binários. Índices podem ser criados para consultas eficientes e operações são realizadas em transações que evita conflitos caso duas abas ou janelas do mesmo aplicativo estejam abertas simultâneamente. É possível criar múltiplos bancos de dados por aplicação, com versionamento, onde mudanças no esquema só são permitidas dentro do tratamento do evento upgradeneeded. O acesso é somente permitido a bancos de dados criados na mesma origem garantindo que bancos de dados de domínios diferentes sejam inacessíveis. Seus objetos são organizados em coleções, denominadas object stores, e são ordenados segundo a chave em ordem ascendente.

O IndexedDB permite busca indexada de alto desempenho utilizando consultas em JavaScript e é útil para armazenamento de grandes quantidades de dados estruturados. Sua API assíncrona é baseada em callbacks, onde eventos são disparados quando a operação é concluída, e também pode ser usada com service workers. É compatível com todos os navegadores modernos alcançando 83% dos usuários. O IndexedDB não foi projetado para ordenação internacionalizada, pode ser realizada fora do banco de dados, sincronização com servidor externo de banco de dados e busca textual pois não possui equivalente ao LIKE do SQL. Navegadores podem apagar o banco de dados em algumas condições, como quando o usuário requisitar, o navegador for utilizado no modo privado ou incógnito ou a quota de disco atingir limite.

Bibliotecas para armazenamento local de terceiros baseadas nas tecnologias web implementadas pelo navegador simplificam a utilização de APIs que em alguns casos são complexas (IndexedDB) ou não possuem suporte em todos os navegadores (Web SQL). Entre elas temos o LocalForage que possui API baseada no localStorage e permite armazenar dados de tipos variados, ao invés de somente

strings. Ele utiliza IndexedDB, WebSQL e localStorage para compatibilidade com navegadores antigos. Outra biblioteca muito utilizada é o PouchDB, que inspirado no CouchDB, facilita desenvolvido de aplicativos offline-first além de permitir sincronização com servidores CouchDB. O PouchDB utiliza IndexedDB e WebSQL para o armazenamento de dados locais.

A biblioteca Dexie.js apresenta uma camada de abstração específica para o IndexedDB com melhor tratamento de erro e consultas simplificadas o que reduz a complexidade do código desenvolvida. Lovefield apresenta um banco de dados relacional com API estilo SQL e armazenamento agnóstico (IndexedDB, Firebase, WebSQL, etc) além de funcionamento multiplataforma, diferente do WebSQL. LokiJS é um banco de dados NoSQL em memória que combina características do Redis, MongoDB e RethinkDB e permite sincronização e persistência de dados e utiliza IndexedDB, localStorage ou sistema de arquivos para o armazenamento de dados locais.

Outra alternativa ao Web SQL é o AlaSQL, um banco de dados SQL em memória que realiza persistência de dados opcional com IndexedDB, localStorage, sistema de arquivos ou SQLite, dependendo do ambiente onde for utilizado. Forerunner apresenta um banco de dados NoSQL para armazenamento de objetos JSON com consultas baseadas no MongoDB e utiliza IndexedDB, WebSQL ou localStorage para armazenamento de dados. E como último exemplo temos o YDN-DB, um banco de dados de objetos com esquema bem-definido e suporte básico a consultas SQL incluindo agregação através de uma camada de processamento JavaScript. Internamento utiliza IndexedDB, WebSQL ou localStorage para o armazenamento de dados.

Referências

2G - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/2G>. Acesso em: 25 de março de 2018.

3G - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/3G>. Acesso em: 25 de março de 2018.

4G - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/4G>. Acesso em: 25 de março de 2018.

A evolução dos dispositivos móveis e a sua influência em nossas vidas | LinkedIn. Disponível em: <https://pt.linkedin.com/pulse/evolu%C3%A7%C3%A3o-dos-dispositivos-m%C3%B3veis-e-sua-influ%C3%AAncia-em-zwierzykowski>. Acesso em: 25 de março de 2018.

A Guide to Mobile App Development: Web vs. Native vs. Hybrid | Clearbridge Mobile. Disponível em: <https://clearbridgemobile.com/mobile-app-development-native-vs-web-vs-hybrid/>. Acesso em: 25 de março de 2018.

A Nossa Tecnologia. Disponível em: <http://getnord.com/index.php/pt/a-nossa-tecnologia>. Acesso em: 25 de março de 2018.

A Powerful Operating System- KaiOS. Disponível em: <https://www.kaiostech.com/>. Acesso em: 25 de março de 2018.

A Real-Time Database Survey: The Architecture of Meteor, RethinkDB, Parse & Firebase. Disponível em: <https://medium.baqend.com/real-time-databases-explained-why-meteor-rethinkdb-parse-and-firebase-dont-scale-822ff87d2f87>. Acesso em: 25 de março de 2018.

About iOS App Architecture. Disponível em: <https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40007072>. Acesso em: 25 de março de 2018.

AlaSQL - javascript SQL database library. Disponível em: <http://alasql.org>. Acesso em: 25 de março de 2018.

Android (operating system) - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Android_(operating_system)>. Acesso em: 25 de março de 2018.

Android Debug Bridge (adb) | Android Studio. Disponível em: <https://developer.android.com/studio/command-line/adb.html>. Acesso em: 25 de março de2018.

Android development is 30% more expensive than iOS. And we have the numbers to prove it! | Infinum. Disponível em: <https://infinum.co/the-capsized-eight/android-development-is-30-percent-more-expensive-than-ios#disqus_thread>. Acesso em: 25 de março de 2018.

Android Open Source Project. Disponível em: <https://source.android.com/>. Acesso em: 25 de março de 2018.

Android Skins Explained: How Do Hardware Makers Change Stock Android?. Disponível em: <https://www.makeuseof.com/tag/android-skins-explained-hardware-makers-change-stock-android/>. Acesso em: 25 de março de 2018.

Android software development - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Android_software_development>. Acesso em: 25 de março de 2018.

Android Studio - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Android_Studio>. Acesso em: 25 de março de 2018.

Android version history - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Android_version_history>. Acesso em: 25 de março de 2018.

Angular (application platform) - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Angular_(application_platform)>. Acesso em: 25 de março de 2018.

Apache CouchDB. Disponível em: <http://couchdb.apache.org/>. Acesso em: 25 de março de 2018.

App Store (iOS) - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/App_Store_(iOS)>. Acesso em: 25 de março de 2018.

App Store Review Guidelines - Apple Developer. Disponível em: <https://developer.apple.com/app-store/review/guidelines/>. Acesso em: 25 de março de2018.

Appcelerator Open Source. Disponível em: <https://www.appcelerator.org/>. Acesso em: 25 de março de 2018.

Apple Original Newton MessagePad Specs @ EveryMac.com. Disponível em: <https://everymac.com/systems/apple/messagepad/stats/newton_mp_omp.html>. Acesso em: 25 de março de 2018.

Application Fundamentals | Android Developers. Disponível em: <https://developer.android.com/guide/components/fundamentals.html>. Acesso em: 25 de março de 2018.

B2C: conceito, desafios e estratégia | Endeavor Brasil. Disponível em: <https://endeavor.org.br/b2c/>. Acesso em: 25 de março de 2018.

Backend for Frontends - a microservices pattern - Technology explained. Disponível em: <https://alexandreesl.com/2016/03/18/backend-for-frontends-a-microservices-pattern/>. Acesso em: 25 de março de 2018.

BFF @SoundCloud | ThoughtWorks. Disponível em: <https://www.thoughtworks.com/insights/blog/bff-soundcloud>. Acesso em: 25 de março de 2018.

BlackBerry 10 - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/BlackBerry_10>. Acesso em: 25 de março de 2018.

Build Amazing Native Apps and Progressive Web Apps with Ionic Framework and Angular. Disponível em: <https://ionicframework.com/>. Acesso em: 25 de março de 2018.

Build WordPress Mobile Apps - Convert any site into mobile apps - MobiLoud Canvas. Disponível em: <https://www.mobiloud.com/wordpress-mobile-app/>. Acesso em: 25 de março de 2018.

Build Your First App | Android Developers. Disponível em: <https://developer.android.com/training/basics/firstapp/index.html>. Acesso em: 25 de março de 2018.

Business To Consumer (B To C). Disponível em: <https://www.investopedia.com/terms/b/btoc.asp>. Acesso em: 25 de março de 2018.

Capítulo 1 - A evolução dos dispositivos | Apostila Desenvolvimento para dispositivos móveis. Disponível em: <https://devnetgomez.gitbooks.io/desenvolvimento-para-dispositivos-moveis-android/a_evolucao_dos_dispositivos.html>. Acesso em: 25 de março de 2018.

Change Streams — MongoDB Manual 3.6. Disponível em: <https://docs.mongodb.com/manual/changeStreams/>. Acesso em: 25 de março de 2018.

Chrome OS - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Chrome_OS>. Acesso em: 25 de março de 2018.

Cinco modelos de negócio para apps | BloomIdea. Disponível em: <https://bloomidea.com/blog/5-modelos-de-negocio-para-apps>. Acesso em: 25 de março de 2018.

Cinco razões pelas quais Serverless vai dominar o mundo - Blog da Kendoo. Disponível em: <http://blog.kendoo.com.br/2017/02/5-razoes-pelas-quais-serverless-vai-dominar-o-mundo/>. Acesso em: 25 de março de 2018.

COLLINS, Lauren; ELLIS, Scott. *Mobile Devices*: Tools and Technologies. 1ª ed. Chapman and Hall / CRC, 2015.

Comercialização de software em plataforma mobile: um estudo de caso aplicado ao Android. Disponível em: <http://repositorio.ufsm.br/bitstream/handle/1/539/Rech_William_Rodrigues_da_Fonseca.pdf?sequence=1>. Acesso em: 25 de março de 2018.

Compras virtuais por meio de dispositivos móveis crescem 75%. Disponível em: <https://www.ecommercebrasil.com.br/noticias/compras-virtuais-por-meio-de-dispositivos-moveis-crescem-75-no-ultimo-trimestre/>. Acesso em: 25 de março de 2018.

Computação móvel, histórico da evolução. Disponível em: <http://grenoble.ime.usp.br/~gold/cursos/2008/movel/mono/HistoricoComputacaoMovel.pdf>. Acesso em: 25 de março de 2018.

Core app quality | Android Developers. Disponível em: <https://developer.android.com/develop/quality-guidelines/core-app-quality.html>. Acesso em: 25 de março de 2018.

Core Data - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Core_Data>. Acesso em: 25 de março de 2018.

Couchbase Mobile. Disponível em: <https://developer.couchbase.com/mobile>. Acesso em: 25 de março de 2018.

CouchDB - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/CouchDB>. Acesso em: 25 de março de 2018.

CSS | MDN. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/CSS>. Acesso em: 25 de março de 2018.

Dashboards | Android Developers. Disponível em: <https://developer.android.com/about/dashboards/index.html>. Acesso em: 25 de março de 2018.

Databases - Xamarin. Disponível em: <https://developer.xamarin.com/recipes/android/data/databases/>. Acesso em: 25 de março de 2018.

Descubra as possibilidades dos dispositivos móveis. Disponível em: <https://learndigital.withgoogle.com/atelierdigitalportugal/lesson/81>. Acesso em: 25 de março de 2018.

Design | Android Developers. Disponível em: <https://developer.android.com/design/index.html>. Acesso em: 25 de março de 2018.

Designing native apps for Android and iOS: key differences and similarities | Cheesecake Labs. Disponível em: <https://cheesecakelabs.com/br/blog/designing-native-apps-for-android-and-ios-key-differences-and-similarities/>. Acesso em: 25 de março de 2018.

Dexie.js - Minimalistic IndexedDB Wrapper. Disponível em: <http://dexie.org/>. Acesso em: 25 de março de 2018.

Dispositivos móveis no mercado de e-commerce brasileiro. Disponível em: <https://www.pagbrasil.com/pt-br/noticias/dispositivos-moveis-brasil/>. Acesso em: 25 de março de 2018.

Dynabook - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Dynabook>. Acesso em: 25 de março de 2018.

Ecma International - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Ecma_International>. Acesso em: 25 de março de 2018.

ESPOSITO, Dino. *Architecting Mobile Solutions for the Enterprise*. 1ª ed. Microsoft Press, 2012.

Firebase - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Firebase>. Acesso em: 25 de março de 2018.

Firefox OS - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Firefox_OS>. Acesso em: 25 de março de 2018.

ForerunnerDB. Disponível em: <http://forerunnerdb.com>. Acesso em: 25 de março de 2018.

Framework7 - Full Featured Mobile HTML Framework For Building iOS & Android Apps. Disponível em: <http://framework7.io/>. Acesso em: 25 de março de2018.

Fujitsu1. Disponível em: <http://www.telestarcorporation.com/fujitsu1.htm>. Acesso em: 25 de março de 2018.

Fundamentos da Computação Móvel. Disponível em: <http://dai.ifma.edu.br/~mlcsilva/aulasprogmov/Aula%202.pdf>. Acesso em: 25 de março de 2018.

Future batteries, coming soon: Charge in seconds, last months and power over the air - Pocket-lint. Disponível em: <https://www.pocket-lint.com/gadgets/news/130380-future-batteries-coming-soon-charge-in-seconds-last-months-and-power-over-the-air>. Acesso em: 25 de março de 2018.

Get Started with Kotlin on Android | Android Developers. Disponível em: <https://developer.android.com/kotlin/get-started.html>. Acesso em: 25 de março de 2018.

Getting Started with Progressive Web Apps | Web | Google Developers. Disponível em: <https://developers.google.com/web/updates/2015/12/getting-started-pwa>. Acesso em: 25 de março de 2018.

GitHub - ccgus/fmdb: A Cocoa / Objective-C wrapper around SQLite. Disponível em: <https://github.com/ccgus/fmdb>. Acesso em: 25 de março de 2018.

Google Fuchsia - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Google_Fuchsia>. Acesso em: 25 de março de 2018.

Google Play - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Google_Play>. Acesso em: 25 de março de 2018.

Grau de proteção IP – Wikipédia, a enciclopédia livre. Disponível em: <https://pt.wikipedia.org/wiki/Grau_de_prote%C3%A7%C3%A3o_IP>. Acesso em: 25 de março de 2018.

Guide to App Architecture | Android Developers. Disponível em: <https://developer.android.com/topic/libraries/architecture/guide.html>. Acesso em: 25 de março de 2018.

How much does it cost to make an app? - App Cost Calculator. Disponível em: <http://howmuchtomakeanapp.com/>. Acesso em: 25 de março de 2018.

How to Use the Backend for Frontend (BFF) Pattern in Your Mobile Application Architecture - Intellectual Apps Ltd. Disponível em: <https://intellectualapps.com/2017/10/31/how-to-use-the-backend-for-frontend-bff-pattern-in-your-mobile-application-architecture/>. Acesso em: 25 de março de 2018.

HTML | MDN. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/HTML>. Acesso em: 25 de março de 2018.

I Want to Write Android Apps. Where Do I Start?. Disponível em: <https://lifehacker.com/i-want-to-write-android-apps-where-do-i-start-1643818268>. Acesso em: 25 de março de 2018.

IDC: Smartphone OS Market Share. Disponível em: <https://www.idc.com/promo/smartphone-market-share/os>. Acesso em: 25 de março de 2018.

IDC: Smartphone Vendor Market Share. Disponível em: <https://www.idc.com/promo/smartphone-market-share/vendor>. Acesso em: 25 de março de 2018.

IndexedDB API - Web APIs | MDN. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API>. Acesso em: 25 de março de 2018.

Introducing Clean Swift Architecture (VIP) – Hacker Noon. Disponível em: <https://hackernoon.com/introducing-clean-swift-architecture-vip-770a639ad7bf>. Acesso em: 25 de março de 2018.

Introduction to Service Worker | Web | Google Developers. Disponível em: <https://developers.google.com/web/ilt/pwa/introduction-to-service-worker>. Acesso em: 25 de março de 2018.

Ionic Native - SQLite. Disponível em: <https://ionicframework.com/docs/native/sqlite/>. Acesso em: 25 de março de 2018.

iOS - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/IOS>. Acesso em: 25 de março de 2018.

iOS SDK - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/IOS_SDK>. Acesso em: 25 de março de 2018.

iOS version history - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/IOS_version_history>. Acesso em: 25 de março de 2018.

IP Code - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/IP_Code>. Acesso em: 25 de março de 2018.

iPad - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/IPad>. Acesso em: 25 de março de 2018.

iPad Pro keyboard and Apple Pencil review: Is this the future of personal computing?. Disponível em: <http://www.ibtimes.co.uk/ipad-pro-keyboard-apple-

pencil-review-this-future-personal-computing-1557522>. Acesso em: 25 de março de 2018.

iPhone - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/IPhone>. Acesso em: 25 de março de 2018.

Java vs. Kotlin: Which is the Better Option for Android App Development? | Clearbridge Mobile. Disponível em: <https://clearbridgemobile.com/java-vs-kotlin-which-is-the-better-option-for-android-app-development/>. Acesso em: 25 de março de 2018.

JavaScript | MDN. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>. Acesso em: 25 de março de 2018.

Jornada (PDA) - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Jornada_(PDA)>. Acesso em: 25 de março de 2018.

KaiOS - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/KaiOS>. Acesso em: 25 de março de 2018.

Kotlin Programming Language. Disponível em: <https://kotlinlang.org/>. Acesso em: 25 de março de 2018.

Learn web development | MDN. Disponível em: <https://developer.mozilla.org/en-US/docs/Learn>. Acesso em: 25 de março de 2018.

LevelDB.org. Disponível em: <http://leveldb.org/>. Acesso em: 25 de março de 2018.

Lightweight javascript in-memory database: LokiJS. Disponível em: <http://lokijs.org>. Acesso em: 25 de março de 2018.

localForage. Disponível em: <https://localforage.github.io/localForage/>. Acesso em: 25 de março de 2018.

Lovefield: The Data Access Library that Loves You. Disponível em: <https://google.github.io/lovefield/>. Acesso em: 25 de março de 2018.

MeeGo - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/MeeGo>. Acesso em: 25 de março de 2018.

Meet Android Studio | Android Studio. Disponível em: <https://developer.android.com/studio/intro/index.html>. Acesso em: 25 de março de 2018.

MessagePad - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/MessagePad>. Acesso em: 25 de março de 2018.

MIT App Inventor | Explore MIT App Inventor. Disponível em: <http://appinventor.mit.edu/explore/index-2.html>. Acesso em: 25 de março de 2018.

Mobile app - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Mobile_app>. Acesso em: 25 de março de 2018.

Mobile app development - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Mobile_app_development>. Acesso em: 25 de março de 2018.

Mobile App Vs. Mobile Website: Which Is The Better Option? | Clearbridge Mobile. Disponível em: <https://clearbridgemobile.com/mobile-app-vs-mobile-website-which-is-the-better-option/>. Acesso em: 25 de março de 2018.

Mobile Database Comparison: SQLite and SQLite alternatives compared in a handy matrix - ObjectBox. Disponível em: <http://objectbox.io/sqlite-alternatives/>. Acesso em: 25 de março de 2018.

Mobile device - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Mobile_device>. Acesso em: 25 de março de 2018.

Mobile devices are the future of work. Disponível em: <https://blog.lookout.com/mobile-devices-future-of-work>. Acesso em: 25 de março de 2018.

Mobile marketing statistics 2018. Disponível em: <https://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/>. Acesso em: 25 de março de 2018.

Mobile operating system - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Mobile_operating_system>. Acesso em: 25 de março de 2018.

Mobile: Native Apps, Web Apps, and Hybrid Apps. Disponível em: <https://www.nngroup.com/articles/mobile-native-apps/>. Acesso em: 25 de março de 2018.

Mobilidade em análise. Disponível em: <https://www.devmedia.com.br/mobilidade-em-analise/3309>. Acesso em: 25 de março de 2018.

MongoDB - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/MongoDB>. Acesso em: 25 de março de 2018.

Motorola DynaTAC - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Motorola_DynaTAC>. Acesso em: 25 de março de 2018.

Native App vs Web App: What's the Difference?. Disponível em: <https://www.upwork.com/hiring/mobile/native-app-vs-web-app-for-mobile/>. Acessoem: 25 de março de 2018.

NativeScript - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/NativeScript>. Acesso em: 25 de março de 2018.

NEIL, Theresa. *Mobile Design Pattern Gallery*. Color Edition. O'Reilly Media, 2012.

Objectbox Mobile Database - ObjectBox. Disponível em: <http://objectbox.io/>. Acesso em: 25 de março de 2018.

Pilot 1000 - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Pilot_1000>. Acesso em: 25 de março de 2018.

PouchDB, the JavaScript Database that Syncs!. Disponível em: <https://pouchdb.com>. Acesso em: 25 de março de 2018.

Progressive Web Apps - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Progressive_web_app>. Acesso em: 25 de março de 2018.

Progressive Web Apps Training | Web | Google Developers. Disponível em: <https://developers.google.com/web/ilt/pwa/>. Acesso em: 25 de março de 2018.

Qual é a diferença entre B2B e B2C? - E-Commerce News. Disponível em: <https://ecommercenews.com.br/artigos/cases/qual-e-a-diferenca-entre-b2b-e-b2c/>. Acesso em: 25 de março de 2018.

React - A JavaScript library for building user interfaces. Disponível em: <https://reactjs.org/>. Acesso em: 25 de março de 2018.

React (JavaScript library) - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/React_(JavaScript_library)>. Acesso em: 25 de março de 2018.

React Native · A framework for building native apps using React. Disponível em: <https://facebook.github.io/react-native/showcase.html>. Acesso em:25 de março de 2018.

React Native at Instagram – Instagram Engineering. Disponível em: <https://engineering.instagram.com/react-native-at-instagram-dd828a9a90c7>. Acesso em: 25 de março de 2018.

Real Time Database Systems. Disponível em: <https://www.cs.helsinki.fi/u/jplindst/papers/rtds.pdf>. Acesso em: 25 de março de 2018.

Realm: Create reactive mobile apps in a fraction of the time. Disponível em: <https://realm.io/>. Acesso em: 25 de março de 2018.

Real-time database - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Real-time_database>. Acesso em: 25 de março de 2018.

Reflexões sobre Arquitetura da Informação para dispositivos móveis. Disponível em: <http://seer.ufrgs.br/index.php/EmQuestao/article/viewFile/55616/37092>. Acesso em: 25 de março de 2018.

Responsive Web Design Basics | Web Fundamentals | Google Developers. Disponível em: <https://developers.google.com/web/fundamentals/design-and-ux/responsive/>. Acesso em: 25 de março de 2018.

RethinkDB: the open-source database for the realtime web. Disponível em: <https://www.rethinkdb.com/>. Acesso em: 25 de março de 2018.

Rugged Phones | Cat phones - Catphones En Gb. Disponível em: <https://www.catphones.com/>. Acesso em: 25 de março de 2018.

Ruggedized handheld device input effectiveness by generation: A time and error study - ScienceDirect. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0169814116300488>. Acesso em: 25 de março de 2018.

Sailfish OS - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Sailfish_OS>. Acesso em: 25 de março de 2018.

Sam Newman - Backends For Frontends. Disponível em: <https://samnewman.io/patterns/architectural/bff/>. Acesso em: 25 de março de 2018.

Save data in a local database using Room | Android Developers. Disponível em: <https://developer.android.com/training/data-storage/room/index.html>. Acesso em: 25 de março de 2018.

Save Data using SQLite | Android Developers. Disponível em: <https://developer.android.com/training/data-storage/sqlite.html>. Acesso em: 25 de março de 2018.

Semantic Web - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Semantic_Web>. Acesso em: 25 de março de 2018.

Serverless computing - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Serverless_computing>. Acesso em: 25 de março de 2018.

Server-side scripting - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Server-side_scripting>. Acesso em: 25 de março de 2018.

Should You Learn Swift or Objective-C? The Short Answer is Both. Disponível em: <http://blog.teamtreehouse.com/learn-swift-or-objective-c>. Acesso em: 25 de março de 2018.

Simulator Overview - Simulator Help. Disponível em: <https://help.apple.com/simulator/mac/current/#/deve44b57b2a>. Acesso em: 25 de março de 2018.

Single-page application - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Single-page_application>. Acesso em: 25 de março de 2018.

SQLite - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/SQLite>. Acesso em: 25 de março de 2018.

SQLite Home Page. Disponível em: <https://sqlite.org>. Acesso em: 25 de março de 2018.

Swift Playgrounds - Apple Developer. Disponível em: <https://developer.apple.com/swift-playgrounds/>. Acesso em: 25 de março de 2018.

Swift Playgrounds - Apple. Disponível em: <https://www.apple.com/swift/playgrounds/>. Acesso em: 25 de março de 2018.

Swift vs Objective-C: What is the Best Language for iOS Development?. Disponível em: <https://www.cleveroad.com/blog/swift-vs-objective-c--what-is-the-best-language-for-ios-development->. Acesso em: 25 de março de 2018.

Swift vs. Objective-C: 10 reasons the future favors Swift | InfoWorld. Disponível em: <https://www.infoworld.com/article/2920333/mobile-development/swift-vs-objective-c-10-reasons-the-future-favors-swift.html>. Acesso em: 25 de março de 2018.

Symbian - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Symbian>. Acesso em: 25 de março de 2018.

Tablet computer - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Tablet_computer>. Acesso em: 25 de março de 2018.

TANENBAUM, Andrew S.; WETHERALL, David. *Redes de Computadores*. Pearson, 2014.

Teaching Activities - Mozilla Learning. Disponível em: <https://learning.mozilla.org/en-US/activities>. Acesso em: 25 de março de 2018.

Teste de computador robusto. Computadores robustos são mais do que apenas “carcaças duras”. Disponível em: <https://www.handheldgroup.com/pt-BR/porque-computadores-robustos/testes-robustos/>. Acesso em: 25 de março de 2018.

The App Shell Model | Web Fundamentals | Google Developers. Disponível em: <https://developers.google.com/web/fundamentals/architecture/app-shell>. Acesso em: 25 de março de 2018.

The Back-end for Front-end Pattern (BFF). Disponível em: <http://philcalcado.com/2015/09/18/the_back_end_for_front_end_pattern_bff.html>. Acesso em: 25 de março de 2018.

The Future is Here: What’s Next For Mobile Phones? | At the Smithsonian | Smithsonian. Disponível em: <https://www.smithsonianmag.com/smithsonian-institution/the-future-is-here-whats-next-for-mobile-phones-180951479/>. Acesso em: 25 de março de 2018.

The future of mobile technology and smartphones | ITProPortal. Disponível em: <https://www.itproportal.com/2016/05/30/the-future-of-mobile-technology-and-smartphones/>. Acesso em: 25 de março de 2018.

The imminent future of smartphones - Handhelds - Budget Smartphones - High-End Smartphones - PC & Tech Authority. Disponível em: <https://www.pcauthority.com.au/feature/the-imminent-future-of-smartphones-485058>. Acesso em: 25 de março de 2018.

The Twelve-Factor App. Disponível em: <https://12factor.net/processes>. Acesso em: 25 de março de 2018.

The Web App Manifest | Web Fundamentals | Google Developers. Disponível em: <https://developers.google.com/web/fundamentals/web-app-manifest/>. Acesso em: 25 de março de 2018.

Themes - Overview - iOS Human Interface Guidelines. Disponível em: <https://developer.apple.com/ios/human-interface-guidelines/overview/themes/>. Acesso em: 25 de março de 2018.

Titanium | Appcelerator Inc. Disponível em: <https://www.appcelerator.com/Titanium/>. Acesso em: 25 de março de 2018.

Tizen - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Tizen>. Acesso em: 25 de março de 2018.

Twitter Lite PWA Significantly Increases Engagement and Reduces Data Usage | Web | Google Developers. Disponível em: <https://developers.google.com/web/showcase/2017/twitter>. Acesso em: 25 de março de 2018.

Ubuntu Touch - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Ubuntu_Touch>. Acesso em: 25 de março de 2018.

UI Design Do’s and Don’ts - Apple Developer. Disponível em: <https://developer.apple.com/design/tips/>. Acesso em: 25 de março de 2018.

Visual Studio Code - Code Editing. Redefined. Disponível em: <https://code.visualstudio.com/>. Acesso em: 25 de março de 2018.

Você sabe o que é a arquitetura em Serverless? - Cloud 21. Disponível em: <https://cloud21.com.br/computacao-em-nuvem/o-que-e-arquitetura-serverless/>. Acesso em: 25 de março de 2018.

Web browser - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Web_browser>. Acesso em: 25 de março de 2018.

Web Storage API - Web APIs | MDN. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API>. Acesso em: 25 de março de 2018.

Web technology for developers | MDN. Disponível em: <https://developer.mozilla.org/bm/docs/Web>. Acesso em: 25 de março de 2018.

Web, Hybrid Or Native Apps? What's The Difference?. Disponível em: <https://www.mobiloud.com/blog/native-web-or-hybrid-apps/>. Acesso em: 25 de março de 2018.

WebOS - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/WebOS>. Acesso em: 25 de março de 2018.

Webpack. Disponível em: <https://webpack.js.org/>. Acesso em: 25 de março de 2018.

What is native app? - Definition from WhatIs.com. Disponível em: <http://searchsoftwarequality.techtarget.com/definition/native-application-native-app>. Acesso em: 25 de março de 2018.

Why "Progressive Web Apps vs. native" is the wrong question to ask. Disponível em: <https://medium.com/dev-channel/why-progressive-web-apps-vs-native-is-the-wrong-question-to-ask-fb8555addcbb>. Acesso em: 25 de março de 2018.

Why Build Progressive Web Apps | Web | Google Developers. Disponível em: <https://developers.google.com/web/ilt/pwa/why-build-pwa>. Acesso em: 25 de março de 2018.

Why Serverless?. Disponível em: <https://serverless.com/learn/>. Acesso em: 25 de março de 2018.

Windows 10 Mobile - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Windows_10_Mobile>. Acesso em: 25 de março de 2018.

Windows Embedded Compact - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Windows_Embedded_Compact>. Acesso em: 25 de março de 2018.

World Wide Web – Wikipédia, a enciclopédia livre. Disponível em: <https://pt.wikipedia.org/wiki/World_Wide_Web>. Acesso em: 25 de março de 2018.

World Wide Web - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/World_Wide_Web>. Acesso em: 25 de março de 2018.

World Wide Web Consortium - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/World_Wide_Web_Consortium>. Acesso em: 25 de março de 2018.

Xcode - Apple Developer. Disponível em: <https://developer.apple.com/xcode/>. Acesso em: 25 de março de 2018.

Xcode - Wikipedia. Disponível em: <https://en.wikipedia.org/wiki/Xcode>. Acesso em: 25 de março de 2018.

YDN-DB javascript library. Disponível em: <http://dev.yathit.com/ydn-db/index.html>. Acesso em: 25 de março de 2018.

Your First Progressive Web App | Web Fundamentals | Google Developers. Disponível em: <https://developers.google.com/web/fundamentals/codelabs/your-first-pwapp/>. Acesso em: 25 de março de 2018.

Zaez - Comunicação Digital Mobile Day: a evolução dos dispositivos móveis. Disponível em: <http://www.zaez.net/mobile-day-a-evolucao-dos-dispositivos-moveis.html>. Acesso em: 25 de março de 2018.