Com o advento das tecnologias, a sociedade passou a ter mais tarefas. A informática se tornou essencial no auxílio dessas tarefas. O formato fixo de um desktop deixou a desejar. A mobilidade se tornou necessária.

Então, o Smartphone surgiu!	
Acesso à internet	***Apps Bancários***
Fotos de alta resolução	***Utilização das nuvens (Cloud)***
Comunicadores	***Jogos On-line***
Redes Sociais	***Streamming***
E-mail	***Realização de Ligações***

Na visão dos dispositivos móveis, a computação móvel teve início em 1992 com o Newton Message PAD, da Apple. Em 1996, a US Robotics lançou o Palm Pilot 1000 e 5000. Ainda em 1996 a Microsoft lançou o Windows CE 1.0, que eram utilizados no NEC MobilePro 200 e no Casio A-10.

Em 1998 a empresa Symbiam, formada pelas empresas Nokia e PSION lançou uma das maiores contribuições para o mercado móvel, o Symbiam OS. Ele possibilitou o surgimento dos Smartphones, sob a conduta da Nokia. Tornou-se o sistema operacional mobile mais vendido no mundo até o final de 2010.

Em 2007 a Apple lançou o iOS, sistema operacional para iPhones, com o primeiro nome de "iPhone OS". Inovou ao trazer as telas sensíveis ao toque e nada de teclado físico. Assim surgiu o touch screen. Este sistema só está disponível para uso na arquitetura dos iPhones, pois possui licença proprietária e torna o seu código fechado.

Ainda em 2007 surgiu o sistema operacional Android, criado por Andy Rubin. Permitiu que os dispositivos se integrassem a apps do Google, como o Maps, YouTube e o Android Market (substituído posteriormente pelo Google Play Store). Como é um S.O. disponibilizado pela Google sob licença de código aberto, muitos fabricantes de smartphones o adotaram.

Segundo Tanenbaum (2016): O Sistema Operacional tem como função fornecer aos programas (softwares) do usuário um modelo de computador melhor, mais simples e mais limpo, assim como lidar com o gerenciamento de todos os recursos disponíveis.

Sem o sistema operacional	***Com o sistema operacional***
O usuário deveria conhecer os detalhes do hardware	Maior racionalidade
Cada hardware possui suas peculiaridades	Mais dedicação aos problemas de alto nível
A complexidade ao se realizar uma tarefa induziria a erros	Portabilidade dos softwares entre diferentes tipos de hardware;
Se tornaria uma tarefa árdua manipular um computador ou dispositivo	Possibilidade de intercomunicação entre os dispositivos
Dificuldade de intercomunicação entre os dispositivos	Abstração do conceito de baixo nível ao usuário

Então concluímos que...

A interação com o Sistema Operacional é de suma importância para o sucesso da tecnologia.

A respeito dos Smartphones, atualmente, temos dois principais concorrentes: que são o IOS da Apple e o Android da Google.

Aplicativos Hibridos	***Aplicativos Nativos***
Um aplicativo híbrido é desenvolvido para ser utilizado em diversas plataformas a partir de um único código fonte	É um aplicativo desenvolvido especificamente para uma plataforma, seja iOS ou Android, que utiliza uma tecnologia específica para a plataforma. Assim, tem acesso a todos os recursos disponíveis que a plataforma oferece de forma simples e sem problemas
Vantagens	***Vantagens***
➥Custo de desenvolvimento menor: você pode ter aplicativos para mais de uma plataforma com apenas um desenvolvimento ➥É perfeito para quem não tem um investimento muito alto e precisa de um aplicativo executando em mais de uma plataforma ➥Facilidade para desenvolvimento e muito valor agregado	➥Acesso a câmeras, galeria, GPS e outros sem ter que fazer nenhuma adaptação para funcionar corretamente; ➥Resolução de problemas se tornam mais fáceis em um aplicativo nativo, pois será levado em consideração tudo sobre aquela plataforma, sem ter que se preocupar com compatibilidade em outra plataforma, suporte, etc. ➥É a melhor opção para aplicativos complexos como os que necessitam de soluções específicas
Desvantagens	***Desvantagens***
➥Limitações para usar alguns recursos: algumas tecnologias híbridas tem algumas limitações para acessar recursos nativos ➥Tratamentos no aplicativo para funcionar corretamente: Algumas funcionalidades atuam de forma diferente ou até não existem em alguma plataforma. Isso deve ser levado em conta para poder realizar o devido tratamento ➥Políticas das lojas: Ao desenvolver um aplicativo híbrido é necessário estar atento às políticas da lojas. Pois podem haver regras diferentes de uma loja para outra	➥Custo de desenvolvimento maior: para duas plataformas o custo é dobrado ➥Encontrar mão de obra qualificada se torna mais caro e difícil ➥Regra de negócio acaba se tornando duplicada: o desenvolvimento da regra de negócio do seu app terá que ser feita nas duas ou mais plataformas que você escolher. Se fosse no híbrido seria desenvolvido apenas uma vez

JAVA

Criada em 1991 pela Sun Microsystems, adquirida pelo Oracle, é uma das linguagens de programação mais utilizadas no mundo, funciona a partir de uma VM, chamada JVM (Java Virtual Machine), É utilizada para desenvolver códigos nativos na plataforma Android.

KOTLIN

• Desenvolvida pela JetBrains em 2010, é uma linguagem relativamente nova em comparação ao Java, Sintaxe limpa e concisa, Programação front-end usando Kotlin/JS, também é utilizada para desenvolvimento nativo no Android.

SWIFT

Surgiu em 2014 criada pela Apple, em 2015 passou a ser Open Source, Simples de utilizar, Moderna e flexível, possui um ótimo desempenho, funciona somente nas plataformas Apple.

Dart

Surgiu em 2011 criada pela Google, Facilidade de aprendizado, ampla documentação, Alta performance, Linguagem padrão do Framework Flutter para híbrido desenvolvimento, adotada para desenvolvimento no Fuchsia, novo S.O. da Google que substituirá o Android.

Javascript

Surgiu em 1995 criada pela extinta Netscape, atualmente conhecida por Mozilla, Alta compatibilidade Mozilla, Alta compatibilidade, atua na dinamicidade e interatividade das páginas, pode ser utilizada em diversos Frameworks como: Ionic, Phonegap, React, Native, eléctron, Flutter e etc.

OBS:

Os slides 2 e 3 que fala sobre a linguagem dart não será necessária pro PDF, pois é só praticar a linguagem... Entramos nos slides 4, 5 e 6:

Stateless widgets X Statefull Widgets

➤O ***Stateless*** não possui estado interno mutável, São construídos apenas uma vez. Uma vez criado, não poderá ser alterado e nem atualizado durante a execução do aplicativo!

➤O ***Statefull*** praticamente o contrário, possui estado interno mutável. Eles podem ser construídos várias vezes durante a execução do aplicativo. Uma vez criado, ele pode ser atualizado em tempo real quando ocorrem mudanças de estado, e a interface do usuário é atualizada de acordo.

Scaffold X Container X Columns X Rows

➤O ***Scaffold*** é um widget fundamental no Flutter que fornece uma estrutura básica para a interface do usuário de um aplicativo. Ele facilita a criação de layouts padronizados, oferecendo suporte a componentes essenciais como AppBar, Drawer, BottomNavigationBar, FloatingActionButton e Body. Cada um desses elementos pode ser personalizado para atender às necessidades do aplicativo

➤Os principais atributos do Scaffold:

➥***appBar***: define a barra de aplicativos na parte superior da tela. Pode conter um título, ações e outros widgets personalizados.

➥***body***: define o conteúdo principal da tela. Pode ser qualquer widget, como um ListView, um GridView, um Container ou um widget personalizado.

➥***floatingActionButton***: define um botão de ação flutuante na tela. Pode ser usado para acionar ações comuns, como adicionar um novo item à lista ou abrir uma tela de configurações.

➥***bottomNavigationBar***: define uma barra de navegação na parte inferior da tela. Pode ser usada para navegar entre as diferentes telas do aplicativo ou para acessar outras funcionalidades.

➥***drawer***: define um menu lateral que pode ser aberto deslizando a tela da esquerda para a direita. Pode ser usado para acessar configurações, opções de ajuda, perfis de usuário ou outras funcionalidades

Child X Children:

➤***Child***: É usado quando um widget pode ter apenas um filho. Assim, o valor esperado para child é um único widget.

➤***Children***: É usado em widgets que podem ter múltiplos filhos. Assim, o valor esperado para children é uma lista de widgets.

➤O ***Container*** é um widget no Flutter que funciona como um bloco visual personalizável. Ele permite ajustar a aparência de outros widgets, oferecendo recursos como cor de fundo, tamanho, margens, padding, bordas e transformações. Versátil, o Container é amplamente utilizado para estilizar e estruturar a interface do usuário.

➤Os principais atributos do container:

➥***alignment***: define a posição do widget filho dentro do Container.

➥***color***: define a cor de fundo do Container.

➥***width***: define a largura do Container.

➥***height***: define a altura do Container.

➥***margin***: define o espaço ao redor do Container.

➥***padding***: define o espaço interno do Container.

➥***decoration***: define a aparência visual do Container. Pode incluir bordas, gradientes e sombras.

➤O ***Column*** é um widget de layout no Flutter que organiza seus filhos verticalmente. Ele é usado para estruturar interfaces em colunas, permitindo alinhar widgets ao topo por padrão. O Column pode conter qualquer tipo de widget e oferece opções de alinhamento e espaçamento para personalização do layout.

➤**Os principais atributos do Column:**

➤***children***: define os widgets filhos do Column. Cada widget filho é empilhado na coluna na ordem em que são definidos.

➤***mainAxisAlignment***: define como os widgets filhos são alinhados verticalmente dentro do Column.

➥***Start*** (alinhado ao topo)

➥***End*** (Alinhado na parte inferior)

➥***Center*** (Alinhado ao centro)

➥***SpaceBetween*** (Espaço igual entre os widgets)

➥***SpaceAround*** (Espaço igual ao redor dos widgets)

➥***SpaceEvenly*** (Espaço igual entre os widgets ao redor deles)

➤***CrossAxisAlignment***: define como os widgets filhos são alinhados horizontalmente dentro do Column.

➥***Start*** (Alinhado a esquerda)

➥***End*** (Alinhado a direita)

➥***Center*** (Alinhado ao centro)

➥***Baseline*** (Alinhado com a linha de base do texto)

➥***Stretch*** (Esticado para preencher todo o espaço disponivel)

➤***MainAxisSize***: define o tamanho principal do Column,

➥***Máximo*** (para preencher todo o espaço disponível)

➥***Minimo*** (para se ajustar ao tamanho dos filhos).

➤O ***Row*** é um widget de layout no Flutter que organiza seus filhos horizontalmente em uma única linha, alinhando-os à esquerda por padrão. Seu alinhamento pode ser personalizado com o atributo MainAxisAlignment, enquanto o CrossAxisAlignment permite ajustar a distribuição dos widgets ao longo do eixo vertical. O Row pode ocupar todo o espaço disponível ou ter um tamanho ajustável conforme necessário.

➤**Os principais atributos do Row:**

➥***Children***: define os widgets filhos do Row. Cada widget filho é disposto em uma linha horizontal na ordem em que são definidos.

➥***MainAxisAlignment***: define como os widgets filhos são alinhados horizontalmente dentro do Row.

➥*Start* (Alinhado á esquerda)

➥*End* (Alinhado a direita)

➥*Center* (Alinhado no centro)

➥*SpaceBetween* (Espaço igual entre os widgets)

➥*SpaceAround* (Espaço igual ao redor dos widgets)

➥*SpaceAround* (Espaço igual entre e ao redor deles)

CrossAxisAlignment: define como os widgets filhos são alinhados verticalmente dentro do Row.

➥*Start* (Alinhado ao topo)

➥*End* (Alinhado na parte inferior)

➥*Center* (Alinhado no centro)

➥*baseline* (Alinhado com a linha de base do texto)

➥*Stretch* (Esticado para preencher todo o espaço disponivel)

mainAxisSize: define o tamanho principal do Row

➥***Máximo*** (para preencher todo o espaço disponível)

➥***Minimo*** (para se ajustar ao tamanho dos filhos).

➤ Importância das Transições de Tela em Aplicativos

A transição de telas em um aplicativo desempenha um papel crucial na experiência do usuário. Algumas razões pelas quais a transição de telas é importante, são:

➥Feedback Visual: As transições indicam que uma ação foi concluída com sucesso, ajudando o usuário a perceber que o aplicativo está respondendo às suas interações.

➥Orientação Espacial: Animações ajudam a manter a noção de navegação, permitindo que o usuário compreenda melhor a hierarquia das telas.

➥Delimitação de Contexto: Cada tela possui um propósito específico, e as transições ajudam a separar claramente diferentes seções do aplicativo.

➥Melhoria da Usabilidade: Navegações suaves e rápidas reduzem a sensação de espera, tornando a experiência mais fluida e responsiva.

➥Efeitos Emocionais: Transições bem projetadas transmitem qualidade, criam uma experiência agradável e aumentam a satisfação do usuário.

➤ Os Widgets de navegação permitem que o usuário mova-se pelo aplicativo ou que este leve a diferentes partes de suas funcionalidades.

➥A classe utilizada para estas transições chama-se Navigator. A Navigator fornece vários métodos de transição, entre os mais utilizados temos o push( ) e o pop( ) para a inclusão ou remoção de rotas.

➥A utilização do Navigator acontece através de widgets clicáveis como: TextButton, ElevatedButton, GestureDetector, BottomNavigationBar e etc.

➥Além disso, ele facilita a passagem de dados entre as telas

➤ Transportar dados entre as telas de um aplicativo é de extrema importância, pois permite a comunicação e o compartilhamento de informações entre os diferentes componentes do aplicativo. Algumas razões pelas quais o transporte de dados entre telas é essencial, são:

➥Compartilhamento de Informações: Permite a transferência de dados entre telas, como credenciais do usuário, preferências, detalhes de produtos, pedidos e outros elementos essenciais para a navegação.

➥Personalização da Experiência: Facilita a exibição de conteúdo adaptado ao usuário, ajustando a interface, configurações e recomendações com base nas informações recebidas.

➥Manutenção do Estado do Aplicativo: Garante a persistência de dados temporários, como filtros aplicados, itens no carrinho de compras e progresso em formulários, proporcionando uma experiência contínua.

➥Separação de Responsabilidades: Mantém cada tela focada apenas em sua funcionalidade específica, promovendo uma arquitetura modular e facilitando a manutenção do código.

➥Integração com Serviços Externos: Permite a comunicação com APIs, possibilitando a recuperação e atualização de informações em tempo real com base nos dados transportados.

➥Para transportar dados usando o MaterialPageRoute no Flutter, você pode utilizar a funcionalidade arguments através da classe RouteSettings.

➥Os argumentos podem ser qualquer tipo de dado, como objetos, listas, mapas, etc.

➥Para receber os dados enviados, utiliza se a classe ModalRoute, que captura os argumentos enviados ao novo contexto e pode atribuí-lo a uma variável, objeto ou lista na tela de destino.