Rafael Perroni

View: View é uma classe fundamental em Android que representa um elemento de interface do usuário. Ela é usada para exibir e interagir com elementos gráficos na tela do dispositivo. a classe View é a base para muitas outras classes de interface do usuário em Android, como Button, TextView, ImageView e EditText. Cada uma dessas classes herda da classe View e adiciona suas próprias funcionalidades e propriedades. a View pode ser configurada com várias propriedades, como tamanho, posição, cor e transparência. Além disso, a View também pode responder a eventos de entrada do usuário, como toques, cliques e gestos.

Layout_width e layout_height: definem a largura e a altura do elemento na tela. –

Background: define a cor de fundo do elemento. –

Text: define o texto a ser exibido no elemento, como um TextView ou Button. –

Src: define a imagem a ser exibida em um ImageView. –

OnClick: define o método a ser chamado quando o elemento for clicado. –

Visibility: define se o elemento é visível ou não na tela. –

Padding: define o espaço em branco em torno do conteúdo do elemento. –

Gravity: define a posição do conteúdo dentro do elemento, como no centro ou à esquerda.

API: Requer o uso da internet para acessar os dados. Suporta uma maior quantidade de dados e consome pouco espaço no dispositivo. As informações são obtidas de um servidor remoto por meio de requisições.

Área de trabalho privada: Cada aplicativo tem sua própria área de trabalho onde os arquivos são armazenados. Eles podem optar por salvar na área pública, caso desejem. Arquivos locais podem ser em formatos como txt (csv, json, xml) ou dat (binário). O acesso aos arquivos locais é mais rápido sem a necessidade de internet, porém ocupa mais espaço em disco e memória. É mais fácil para leitura, relacionamento e atualização dos dados.

SQLite: É um banco de dados embutido que suporta uma maior quantidade de dados. Ele oferece facilidade na localização de dados e nas operações de inserção, atualização e exclusão. No entanto, a leitura pode ser mais lenta e o relacionamento entre tabelas pode ser mais complexo. É necessário escrever mais código para utilizar o SQLite.

Shared Preference: É uma forma de armazenamento para gravar chave-valor. Pode ser usado para gravar configurações locais, preferências do usuário ou como cache. Quando o aplicativo é reinstalado, as informações armazenadas nas shared preferences são recriadas. É fácil de usar e não requer configuração adicional.

Servidores HTTP: Apache, Tomcat, Nginx e IIS. Esses servidores são responsáveis por receber as requisições dos dispositivos móveis e fornecer as respostas adequadas.

Banco de Dados Relacional: os bancos de dados relacionais, como MySQL, SQL Server e Postgres, foram mencionados como opções para armazenar dados estruturados e relacionais em aplicativos móveis.

Banco de Dados NoSQL: são flexíveis e escaláveis para lidar com dados não estruturados ou semi-estruturados.

Comunicação do servidor de aplicação: a comunicação entre o dispositivo móvel e o servidor de aplicação pode ser feita em conjunto ou separadamente. Isso significa que a lógica do servidor pode estar no mesmo local físico do aplicativo ou em um servidor remoto.

Arquitetura Orientada a Serviço: a comunicação entre o dispositivo móvel e o servidor de aplicação é feita por meio da arquitetura Orientada a Serviço. Essa arquitetura envolve a disponibilização de serviços específicos que podem ser consumidos pelos dispositivos móveis.

SOAP (Simple Object Access Protocol): é um protocolo que utiliza XML sobre HTTP para a comunicação entre o dispositivo móvel e o servidor. Esse protocolo é usado para trocar informações estruturadas entre diferentes sistemas.

XML (Extensible Markup Language): é utilizado como uma representação de documento semi-estruturado desde 1996. Ele é composto por elementos e atributos que permitem a organização e descrição dos dados.

REST (Representational State Transfer): se baseia no uso adequado dos métodos HTTP (GET, POST, PUT, DELETE) para permitir a interação entre dispositivos móveis e servidores.

Clientes magros: clientes que possuem apenas uma camada e não apresentam código de aplicação personalizado. Esses clientes são totalmente dependentes do servidor e geralmente são utilizados em navegadores web.

Clientes gordos: três camadas: interface com usuário, lógica de negócio e acesso a dados. A comunicação entre esses clientes e o servidor é baixa. Um exemplo de cliente gordo é o WhatsApp.

Nativo: possui alto desempenho e utiliza o ecossistema da plataforma. No entanto, requer um esforço maior no desenvolvimento para garantir o desempenho adequado.

Compile-to-native: um ambiente de terceiros é utilizado para compilar a aplicação para diversas plataformas. Exemplos de frameworks incluem React Native, Native Script e Xamarin. No entanto, dominar esses frameworks pode apresentar certa dificuldade.

Híbrida: fáceis para desenvolvedores web, pois executam em uma webview. No entanto, podem ser mais lentas em comparação com as aplicações nativas. Exemplos de frameworks híbridos incluem PhoneGap, Cordova e Ionic.

Progressive Web App: aplicações são fáceis de desenvolver e não são aplicativos reais, mas executam no navegador. Elas oferecem uma experiência semelhante à de um aplicativo nativo, mas não requerem instalação.

Volley: É uma biblioteca do Google utilizada para fazer chamadas de API. No entanto, é importante ressaltar que não pode ser chamada na thread principal, pois pode bloquear a interface do usuário.

Atualização do SDK: A cada ano, é necessário fazer a atualização do SDK (Software Development Kit) para se manter atualizado com as novas funcionalidades e correções oferecidas pela plataforma.

Classe anônima: Se a chamada à API for utilizada poucas vezes, é possível gerar uma classe anônima para lidar com o callback. Isso evita a necessidade de criar uma classe separada para cada chamada de API.

Dinâmica das APIs: Cada API possui sua própria dinâmica e funciona de forma diferente. Isso significa que cada API pode ter seus próprios métodos e parâmetros específicos, requerendo um mapeamento adequado para a sua utilização.

Componentes e classes: Todos os componentes necessários para a resposta da API devem estar presentes na classe correspondente. Isso significa que os dados retornados pela API devem ser mapeados adequadamente para os campos e propriedades da classe.

Callback: O callback é um mecanismo utilizado para receber uma chamada de retorno com a resposta da API. Esse callback é executado quando a resposta é recebida.

Callback na API Service: O callback deve ser implementado na classe de serviço da API. Essa classe é responsável por fazer a chamada à API e receber a resposta.

OnResponse: O método OnResponse é acionado quando a resposta da API é bemsucedida. Nesse momento, é possível acessar e manipular os dados retornados pela API.

OnFailed: O método OnFailed é acionado quando a resposta da API resulta em falha. Nesse caso, é possível tratar o erro e realizar ações apropriadas, como exibir mensagens de erro ao usuário.