1. Arquitetura MVC com React Native

Este módulo foca na utilização de **arquiteturas robustas**, em particular o padrão arquitetural **Model-View-Controller (MVC)**, que é uma referência no mercado para aplicações cadastrais com uso de bancos de dados.

• **Padrões de Projeto (Design Patterns):** Surgiram para catalogar soluções para problemas recorrentes em desenvolvimento de software, como o GoF (Gang of Four). Exemplos relevantes incluem **Singleton** (define apenas uma instância para uma classe), **Facade** (simplifica subsistemas complexos), **Composite** (utilizado na criação de interfaces hierárquicas no React Native), e **Template Method** (define processos genéricos com lacunas preenchidas por descendentes, como visto no GenericDAO).

• **Camadas MVC:** O padrão promove a divisão do sistema em três camadas:

    1. **Model:** Elementos voltados para a persistência de dados. É a única camada que permite o acesso ao banco de dados. Contém componentes como **DAO (Data Access Object)**, que concentra as operações de acesso ao banco e efetua o mapeamento objeto-relacional.

    2. **Controller:** Componentes que definem o comportamento e as **regras de negócio** do sistema. Apenas o Controller pode invocar os componentes DAO oferecidos pela Model. O GenericController implementa variações do padrão Template Method.

    3. **View:** Telas e interfaces do sistema. Exibe informações obtidas do Controller e envia solicitações do usuário a ele.

--------------------------------------------------------------------------------

2. Arquiteturas Flux e Redux com React Native

Este módulo aborda arquiteturas direcionadas para o **fluxo de dados**, que são alternativas ao MVC, oferecendo um fluxo unidirecional de execução.

Arquitetura Flux

• **Propósito:** Arquitetura utilizada pelo Facebook para um **fluxo unidirecional de dados**. É escalonável e elimina relacionamentos bidirecionais (comuns no MVC).

• **Componentes Principais:**

    1. **View:** Telas do React Native, iniciam uma ação (Action).

    2. **Action:** Padroniza a comunicação entre as telas e o Dispatcher, enviando mensagens.

    3. **Dispatcher:** Canal central que **coordena e repassa as solicitações (mensagens) para a classe Store**.

    4. **Store:** Contém os dados e a **lógica de negócios** da aplicação. Utiliza o padrão **Observer** para notificar automaticamente todas as telas associadas quando os dados são atualizados.

Arquitetura Redux

• **Propósito:** Alternativa ao Flux, seguindo princípios equivalentes, mas com uma **abordagem funcional** e implementação simplificada.

• **Diferença-Chave:** No Redux, **não há o componente Dispatcher** nem a necessidade de eventos personalizados.

• **Mecanismo:** Gerencia o estado do aplicativo de forma centralizada por uma única **árvore de estados imutável**.

• **Manipulação de Estado:** Ocorre por meio de **funções puras** classificadas como **Reducer**. O Reducer funciona como um filtro de ações, processa a solicitação (baseada no type da ação), modifica os dados e retorna o estado alterado.

• **Fluxo de Comunicação:** A View despacha uma Action (função que retorna um objeto com type e payload). O Store notifica mudanças aos componentes visuais associados via **assinatura (subscribe)**.

--------------------------------------------------------------------------------

3. Criptografia com React Native

Este módulo aborda o sigilo das informações e o controle de acesso, utilizando diferentes tipos de criptografia.

• **Conceitos Fundamentais:**

    ◦ **Codificação:** Transformação reversível sem a necessidade de uma chave (ex: Base64).

    ◦ **Criptografia:** Transformação que exige uma **chave** para garantir o sigilo e reverter o processo.

• **Tipos de Criptografia:**

    1. **Simétrica:** Utiliza a **mesma chave (Secret Key)** para criptografia e decriptografia (ex: 3DES, AES, RC4).

    2. **Assimétrica:** Utiliza um par de chaves (Chave Privada para armazenamento seguro e Chave Pública distribuída).

    3. **Destrutiva (Hash):** Ocorre a perda de fragmentos dos dados originais, impedindo a decriptografia; útil para **guarda de senhas** (ex: MD5, SHA1, SHA512).

• **Implementação no React Native:**

    ◦ **Biblioteca react-native-crypto-js:** Disponibiliza algoritmos de hash (MD5, SHA1) e criptografia simétrica (AES, 3DES, RC4).

    ◦ **EncryptedStorage:** Funciona de forma similar ao AsyncStorage, mas armazena valores de forma transparente e **criptografada**, usando as chaves da Key Store nativa do dispositivo (EncryptedSharedPreferences no Android, Keychain no iOS).

    ◦ **Geração de Chaves:** A chave simétrica é gerada com boa randomização, utilizando um **Vetor de Inicialização (IV)** e uma sequência de bytes aleatórios (**SALT**).

    ◦ **Criptografia Destrutiva no Expo:** A biblioteca **expo-crypto** simplifica a geração de hash, como o SHA512, de forma assíncrona, usando o comando digestStringAsync.

--------------------------------------------------------------------------------

4. Publicação de aplicativos com React Native

Este módulo cobre a garantia de qualidade por meio de testes, análise de performance e o processo de publicação.

Teste de Software

• **Objetivo:** Garantir a qualidade, verificar processos pontuais e o efeito de mudanças, e simular situações de erro.

• **Níveis de Teste:** Unitário, de Integração, Sistêmico e de Aceitação.

• **Jest:** É o **framework padrão para testes unitários** no React Native.

    ◦ Os testes são definidos com test ou it e o resultado é verificado com expect usando operadores como toBe ou toEqual.

    ◦ **Mocking:** Utiliza **objetos simulados (mocks)**, como em jest.mock, para fornecer respostas predefinidas, eliminando a necessidade de componentes externos (ex: servidores HTTP) para testes sistêmicos.

    ◦ **Teste de Interface (UI):** Pode ser feito pelo react-test-renderer e **Snapshot Testing** (toMatchInlineSnapshot), que compara o estado atual do componente com um "instantâneo" JSON salvo, detectando qualquer modificação na tela.

Análise de Performance (Performance Tuning)

• **Meta:** Garantir que a interface gráfica seja fluida, trabalhando a pelo menos **60 FPS** (Quadros por Segundo).

• **Ferramentas:** O **DevTools**, na opção Desempenho, permite gravar sequências de ações em modo de depuração para analisar o consumo de memória (Heap JS, GPU) e o tempo de execução de funções (Debugger Worker). Na opção Memória, é possível tirar **instantâneos** de alocação e aplicar filtros para classes específicas.

• **Otimizações:** Usar componentes otimizados (ex: FlatList para grandes massas de dados), diminuir o redesenho de telas (usando React.memo para memorizar componentes), retirar mensagens de log (via plugin transform-remove-console), e substituir o motor JavaScript padrão pelo **Hermes** (que diminui o tempo de carga e o tamanho da biblioteca).

Shipping (Publicação)

• **Processo:** Envolve o empacotamento e o cumprimento dos requisitos das lojas (Apple App Store e Google Play Store).

• **Assinatura Digital:** Exigência comum. Para Android, é necessário criar um certificado digital (.keystore) usando o keytool do JDK. Para publicação, o projeto deve gerar um arquivo **AAB (Android Application Bundle)**.

• **Ambiente Expo:** Simplifica a publicação, oferecendo a opção build (expo build:android ou expo build:ios), que **assina automaticamente o pacote**, dispensando a geração manual da keystore.

--------------------------------------------------------------------------------

5. Conclusão

Os conhecimentos adquiridos neste tema, considerados avançados em React Native, abrangem arquiteturas populares (MVC, Flux, Redux), a implementação de padrões de desenvolvimento e a aplicação de técnicas de garantia de qualidade (testes, análise de performance) e sigilo de informações (criptografia), visando o desenvolvimento de **aplicativos comerciais robustos**.

Exercícios:







