

Avaliação-07

Aluno: Gabriel Souza de Alencar - P5 Informática

Professor: Ricardo Duarte Taveira

Descrição:

Criar um NotebookLM sobre o tema NETWORKING citado em

<https://docs.flutter.dev/data-and-backend/networking>.

Usar como fonte o link acima, videos no youtube relacionados ao tema, e PDF.

Criar um questionário com 20 questões (múltipla escolha 4 opções de resposta) no final relacionar as respostas.

Evidenciar no Github a avaliação-07 com prints da tela do Notebooklm com as fontes, e o questionário em PDF com as respostas.

Questionário:

1. Qual a principal vantagem do pacote **Dio** sobre o pacote **http** para requisições HTTP em Flutter, especialmente em aplicações complexas?
 - A) Ser mais leve e ter menor tamanho de aplicativo.
 - B) **Oferecer decodificação JSON automática e interceptores.**
 - C) Não precisar de adição de dependências no **pubspec.yaml**.
 - D) Ser a única opção para requisições GET e POST.

2. O que o **Dio** permite configurar globalmente que o **http** exige configuração manual em cada requisição?
 - A) Apenas o corpo da requisição (body).
 - B) Somente o tipo de método HTTP (GET, POST, etc.).
 - C) **URL base, cabeçalhos padrão e timeouts de conexão.**
 - D) O número de tentativas de requisição sem conexão.

3. Qual recurso do **Dio** é especialmente útil para adicionar tokens de autenticação automaticamente, registrar requisições/respostas para depuração ou lidar com códigos de status específicos globalmente?
 - A) **CancelToken**

- B) `FormData`
- C) **Interceptores**
- D) `StreamSink`

4. Para upload de arquivos com progresso em Flutter, qual pacote oferece suporte embutido mais simplificado?

- A) Apenas `http` via `MultipartRequest`.
- B) **`Dio` com sua API `FormData` e callbacks de progresso.**
- C) Ambos os pacotes exigem implementações complexas.
- D) Nenhum dos pacotes suporta upload de arquivos.

5. Segundo as fontes, qual é a melhor prática para lidar com a conexão de internet em uma aplicação Flutter antes de fazer uma requisição de rede?

- A) Enviar a requisição e capturar o erro se falhar.
- B) Usar um `try-catch` em cada requisição.
- C) **Verificar a conexão de internet do usuário antes de enviar a requisição.**
- D) Ignorar a conexão, pois o sistema operacional gerencia isso.

6. Ao verificar a conexão de internet em Flutter usando o pacote `connectivity_plus`, qual método é recomendado para garantir que a internet está "realmente" funcionando, além de apenas estar conectado a Wi-Fi ou dados móveis?

- A) Verificar apenas se o dispositivo está conectado ao Wi-Fi.
- B) Tentar enviar uma pequena quantidade de dados para o servidor.
- C) **Tentar fazer um "lookup" para um site conhecido como `google.com`.**
- D) Confiar na notificação do sistema operacional sobre a conexão.

7. Qual é o benefício de criar uma classe auxiliar reutilizável (como `SafeUntap`) para verificar a conexão de internet em uma aplicação

Flutter?

- A) Aumentar a lentidão do aplicativo.
 - B) Duplicar a lógica de verificação em vários lugares.
 - C) **Tornar o código mais limpo e o aplicativo mais inteligente, evitando requisições falhas.**
 - D) Apenas exibir uma mensagem de erro genérica.
8. Qual é uma das vulnerabilidades mais perigosas em aplicativos Flutter, que pode levar ao acesso não autorizado a recursos sensíveis se as chaves forem expostas?
- A) Usar cores de UI inconsistentes.
 - B) **Expor chaves de API (API keys) ou segredos no código-fonte.**
 - C) Não usar `ListView.builder`.
 - D) Não adicionar comentários no código.
9. Para proteger a comunicação entre um aplicativo Flutter e o servidor contra ataques Man-in-the-Middle (MITM), qual protocolo deve ser sempre usado?
- A) HTTP
 - B) FTP
 - C) SMTP
 - D) **HTTPS**
10. Qual técnica de segurança garante que um aplicativo Flutter se comunica apenas com um servidor confiável, comparando o certificado do servidor com um certificado ou chave pública pré-armazenada?
- A) Cache de dados.
 - B) Validação de formulário.
 - C) **Certificate Pinning.**
 - D) Uso de variáveis de ambiente.

11. Como o **Firestore Remote Config** pode ser usado para aumentar a segurança de chaves de API em um aplicativo Flutter?

- A) Armazenando as chaves diretamente no código-fonte do aplicativo.
- B) **Criptografando as chaves e armazenando-as localmente após recuperá-las de forma segura do Firestore.**
- C) Enviar as chaves por e-mail para o usuário.
- D) Exibir as chaves na tela para o usuário verificar.

12. O que a "obfuscação de código Dart" faz para aumentar a segurança de um aplicativo Flutter?

- A) Torna o código mais fácil de ler para outros desenvolvedores.
- B) **Renomeia classes, métodos e variáveis para nomes sem sentido, dificultando a engenharia reversa.**
- C) Reduz o tamanho do arquivo APK sem alterar o código.
- D) Adiciona comentários de segurança ao código.

13. Qual é a principal característica das **WebSockets** que as diferencia das requisições HTTP normais em Flutter?

- A) Elas são usadas apenas para download de arquivos.
- B) Permitem comunicação unidirecional do cliente para o servidor.
- C) **Permitem comunicação bidirecional com um servidor sem polling.**
- D) São exclusivas para envio de dados, não para recebimento.

14. Em Flutter, qual pacote é recomendado para conectar e comunicar-se com um servidor WebSocket?

- A) **http**
- B) **dio**
- C) **web_socket_channel**
- D) **connectivity_plus**

15. Qual é um dos principais desafios ao tentar implementar a sincronização offline de dados em aplicativos Flutter, especialmente

quando o aplicativo está em estado "killed" (encerrado) ou em segundo plano?

- A) Dificuldade em desenhar a UI quando offline.
- B) **Confiabilidade limitada de tarefas em segundo plano devido a otimizações de bateria de fabricantes de dispositivos Android.**
- C) O tamanho do banco de dados local.
- D) A necessidade de múltiplos modelos de dados.

16. Para superar a falta de confiabilidade das tarefas em segundo plano (como **WorkManager**) para sincronização offline, qual solução é sugerida nas fontes, especialmente para aplicativos de mensagens como WhatsApp?

- A) Usar um serviço em primeiro plano (foreground service) constantemente.
- B) Armazenar os dados localmente e sincronizá-los apenas quando o aplicativo for reaberto e tiver internet.
- C) **Enviar notificações push periódicas para acionar a sincronização.**
- D) Aumentar o tempo limite das requisições.

17. Qual pacote de banco de dados local é mencionado nas fontes como sendo utilizado para armazenar dados temporariamente para suporte offline em um aplicativo Flutter de gerenciamento de tarefas?

- A) SQLite
- B) Realm
- C) **Hive**
- D) Firebase Firestore

18. Para consumir dados de uma API REST em Flutter, quais formatos de dados são mais comuns, segundo as fontes?

- A) HTML e CSS.
- B) XML e CSV.
- C) **JSON e XML.**
- D) Texto simples e binário.

19. Qual ferramenta é recomendada para testar APIs e ver como elas retornam as informações antes de implementá-las no Flutter?

- A) Visual Studio Code.
- B) Android Studio.
- C) **Postman.**
- D) DartPad.

20. Em Flutter, ao fazer chamadas API assíncronas e exibir os dados na UI, qual widget é frequentemente utilizado para gerenciar e reagir a estados de **Future** (como carregamento, dados disponíveis ou erro)?

- A) **StatelessWidget**
- B) **Container**
- C) **FutureBuilder**
- D) **Column**

Gabarito:

01. B)

02. C)

03. C)

04. B)

05. C)

06. C)

07. C)

08. B)

09. D)

10. C)

11. B)

12. B)

13. C)

14. C)

15. B)

16. C)

17. C)

18. C)

19. C)

20. C)

Relação das Respostas sobre Networking com Flutter

As questões e suas respostas abordam os principais aspectos do desenvolvimento de aplicações Flutter com funcionalidades de rede, destacando as **ferramentas, técnicas e melhores práticas** para garantir a **eficiência, segurança e usabilidade** das interações com APIs e serviços externos.

- **Escolha de Pacotes HTTP (Dio vs. http):** As questões 1 a 4 ilustram as diferenças fundamentais entre os pacotes **Dio** e **http**. O **Dio** é preferível para projetos maiores e mais complexos devido à sua riqueza de recursos, como **decodificação JSON automática, configuração global, interceptores** para manipulação de requisições/respostas, e **suporte simplificado para upload/download de arquivos com progresso**. Enquanto **http** é mais leve e adequado para casos de uso simples.
- **Verificação de Conectividade de Internet:** As questões 5 a 7 enfatizam a importância de **verificar proativamente a conexão de internet** antes de fazer requisições. Isso melhora a experiência do usuário, evita desperdício de recursos e torna o aplicativo mais inteligente. É crucial não apenas verificar a conectividade básica (Wi-Fi/dados móveis), mas também se há **internet "real" disponível**, o que pode ser feito tentando um "lookup" para um domínio conhecido como **google.com**. A criação de uma **classe auxiliar reutilizável** para essa verificação (**SafeUntap**) centraliza a lógica e torna o

código mais limpo.

- **Segurança de API:** As questões 8 a 12 focam na **segurança das interações com APIs**, um aspecto crítico em qualquer aplicação móvel. Pontos importantes incluem **nunca expor segredos ou chaves de API no código-fonte** (hardcoding), sempre usar **HTTPS para comunicação criptografada**, e implementar técnicas avançadas como **Certificate Pinning** para garantir comunicação apenas com servidores confiáveis. O uso de **Firestore Remote Config** combinado com criptografia e **armazenamento seguro local (flutter_secure_storage)** é uma estratégia robusta para gerenciar chaves de API. A **obfuscação do código Dart** também é uma medida de segurança para dificultar a engenharia reversa.
- **Comunicação com WebSockets:** As questões 13 e 14 introduzem as **WebSockets** como um método de **comunicação bidirecional em tempo real**. O pacote **web_socket_channel** é a ferramenta recomendada em Flutter para implementar essa funcionalidade, permitindo tanto ouvir quanto enviar mensagens de forma contínua.
- **Sincronização Offline:** As questões 15 a 17 abordam os desafios de manter dados sincronizados quando o aplicativo está offline ou em segundo plano. Um dos maiores obstáculos é a **falta de confiabilidade das tarefas em segundo plano** em diversas plataformas Android devido a otimizações de bateria. Para superar isso, **notificações push periódicas** são sugeridas como uma alternativa mais confiável para acionar a sincronização, semelhante ao que aplicativos como WhatsApp fazem. O armazenamento local de dados, utilizando pacotes como **Hive**, é fundamental para o suporte offline.
- **Fundamentos de API REST e UI:** As questões 18 a 20 cobrem conceitos gerais essenciais para trabalhar com APIs REST. O **JSON** é o formato de dados mais comum para intercâmbio com APIs. Ferramentas como **Postman** são **indispensáveis para testar APIs** antes da implementação no código. Finalmente, para apresentar dados assíncronos na interface do usuário em Flutter, o **FutureBuilder** é um **widget poderoso** que lida com os diferentes estados de um **Future** (carregando, dados disponíveis, erro), facilitando a construção de UIs responsivas.