Problem 1: The Flickering UI Glitch (10 Marks)

Scenario:

A Flutter-based mobile application frequently encounters a strange issue—UI elements flicker or display outdated data when multiple users interact with real-time features. The app uses a combination of Firebase Firestore and local caching for performance optimization.

The culprit? A race condition where multiple asynchronous operations (such as Firestore reads/writes and local cache updates) execute concurrently without proper synchronization. As a result, UI elements sometimes show stale data, or certain updates appear twice, leading to an inconsistent user experience.

Questions:

Problem Understanding (3 Marks)

1. What concurrency issue is present in this scenario? Explain how it leads to UI flickering or outdated data display.

Solution Approach (4 Marks)

2. Rewrite the `updateUI` method using a thread-safe approach to ensure data consistency across Firestore and local cache.
3. How would using a state management solution (e.g., Riverpod, Bloc, or Provider) help in preventing race conditions and ensuring UI stability?

Performance Considerations (3 Marks)

4. Using `setState()` in multiple asynchronous callbacks ensures immediate updates but can degrade performance. How does using `StreamBuilder` or `ValueNotifier` compare in terms of efficiency?
5. What trade-offs arise if updates are queued using an event-driven approach (e.g., Firebase Cloud Functions) instead of handling them directly in the Flutter app?

Problem 3: The Lost Notifications Bug (10 Marks)

Scenario:

A Flutter messaging app occasionally faces a peculiar issue—some users don't receive push notifications, while others get duplicate alerts. The app relies on Firebase Cloud Messaging (FCM) to deliver real-time notifications and uses local storage to track received messages.

The culprit? A race condition occurs when multiple background isolates attempt to process incoming notifications simultaneously. Since there's no proper synchronization between FCM handlers and local database writes, some notifications are lost, while others are duplicated.

Questions:

Problem Understanding (3 Marks)

1. What concurrency issue is occurring in this scenario? Explain how it leads to missing or duplicated notifications.

Solution Approach (4 Marks)

2. Rewrite the `handleNotification` method using an atomic operation to ensure each notification is processed exactly once.
3. How would implementing a distributed queue (e.g., Firebase Firestore with Firestore Triggers) help prevent race conditions in handling notifications?

Performance Considerations (3 Marks)

4. Using a global lock (e.g., `Mutex` in Dart) prevents duplicate notifications but may introduce delays. How does using `WorkManager` or `Isolate.spawn()` compare in terms of performance?
5. What trade-offs arise if notifications are first stored in Firestore and then pulled by the app instead of using push-based FCM notifications?