Vorname	Nachname	Beruf	Dauer
Marten	Meißner	Fachinformatiker AE	07.22-07.24

Wochenbericht KW 35. (2023.08.28. - 09.03.)

Für unser aktuelles Projekt musste ich mir eine Lösung ausdenken, wie ich regelmäßig im Hintergrund die App Daten abfrage und aktualisieren kann. Das Feature soll folgendes mache:

- in einem gegebenen Intervall soll es alle Termine abfragen, ob es eine Benachrichtigung erstellen soll.
- In einem anderen Zyklus soll es die Daten erneuern und dafür mit einer GET Methode E-Mail-Adressen Daten abfragen.
- In dem Intervall sollen vorhandene Termine aktualisiert werden können.

Da der Nutzer diese Aufgaben nicht aktiv abarbeiten muss, brauche ich eine Logik, die im Hintergrund läuft oder in einem gegebenen Intervall aktiv wird.
Deswegen beschäftigte ich mich, mit Background-Loops und anderen Logiken, die im Hintergrund arbeiten können, ohne die App zu beeinflussen.
Auf der Offiziellen Dart Dokumentation bin ich über den Begriff Isolate gestolpert. Isolates, werden in Dart Prozess Queues bezeichnet, also welche Aufgaben mit einem Prozessor abgearbeitet werden.
Dart ist von Natur aus nicht Multithreated, das bedeutet Dart arbeitet, linear einen Aufgabenstapel (Queue) ab, wenn also eine Schleife 300 Sekunden zählen lassen würde, um ein Fünfminutenintervall zu bekommen würde die ganze App fünf Minuten warten, bis es die eine Aufgabe abgearbeitet hat.
Um aber den einen Isolate nicht zu blockieren, den ich mit meiner void main() Funktion starte, nicht zu blockieren könnte ich eine Asynchrone Funktion benutzen.
Durch das Keyword „async“ kann ich dem Compiler mitteilen, dass die markierte Funktion zur Laufzeit nicht fertig werden kann und den Befehl der mit „await“ markiert wird, warten müsste, bzw. durch die await Markierung einfach wieder hinten in den Queue kommt und im nächsten Durchlauf wieder abgefragt wird.
Der Rückgabe werd einer Asynchronen Funktion wird dann mit dem Typen Future gekennzeichnet, da es beim ersten Aufrufen der Funktion noch nicht fertig ist, aber in Zukunft ein Ergebnis zurückgeben wird, wie z.b. Future.
Aber alleine, mit einer Schleife oder einem Future Rückgabewert, konnte ich die Aufgabe nicht lösen.
Da Dart standardmäßig als Single Thread Programm ausgeführt wird, heißt es aber nicht, dass es auch nicht Multithread kompatible wäre.
Aber für eine Abfrage, alle paar Minuten einen ganzen Prozessor Thread zu blockieren, wäre auch etwas überdimensioniert.
Nachdem ich also weiter in der Dart Dokumentation geforscht hatte, stolperte ich über die Stream-Klasse.
Streams sind ähnlich wie Future Objekte zur Laufzeit noch unbekannt, werden aber anders, als bei einem Future Objekt immer wieder erwartet.
Da Streams auch Asynchron funktionieren werden in einem Intervall Daten abgefragt und bei jedem Intervall Durchlauf abgefangen, bearbeitet und ausgegeben.
In der Stream-Klasse gibt es die Methode listen(), die listen Methode, ist ähnlich wie beim Musik hören, wartet listen auf die Daten von dem Stream Objekt.
Mit jedem neuen Intervall werden die alten Daten überschrieben und ein Event ausgelöst.

In diesem Event kann ich meine Background-Logik implementieren und den Intervall des Streams nutzen, neue BLoC Events zu aktivieren.
Um mein eigenes Stream-Objekt zu schreiben, benötigte ich noch die Timer Klasse.
Die Timer Klasse zählt wie ein klassischer Countdown einfach die Zeit herunter und startet ein Event, wenn es bei null ankommt. Eine Methode der Timer klasse ist periodic, diese Aktiviert nicht nur ein Event, sondern ruft sich immer wieder Selber auf.
Als Lösung für meine Logik nutze ich also eine selbst geschrieben Stream die nach jedem Timer Intervall meine BloC Events starteten, dadurch muss ich nicht einen weiteren Prozessor Queue Blockieren.
Mein Stream ist dadurch Event getrieben und Asynchron, wird der void main Isolate nicht völlig überfordert.
Schluss endlich war dann meine Aufgabe sehr leicht zu beenden, da, mein Stream minütlich ein Event aktiviert, dem ich nur noch meine BLoC Events übergeben brauchte.

Kontrolliert am: _____ Unterschrift :_____