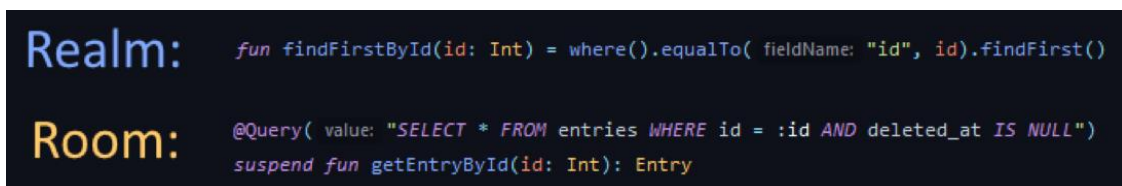


Realisatie stageopdracht 2: Room onderzoek

In het begin van mijn stage werd mij al snel duidelijk gemaakt dat het bedrijf altijd opzoek is naar nieuwe technologieën om mee aan de slag te gaan en zo hun ontwikkelingsproces te stroomlijnen. Mijn stagementor Jesse had mij voor ik begon aan mijn eerste opdracht al verteld over *Room*, een alternatief voor de *Realm library* die ontwikkeld was via *Android Jetpack*. Jesse vertelde me over de verschillen tussen de *libraries* en dat ze intern al hadden nagedacht om over te schakelen naar *Room*. De bedoeling was om mij, nadat ik mij had ingewerkt in de werkwijze van het bedrijf, onderzoek te laten doen naar *Room* en dit te proberen implementeren in de *Overtime* applicatie.

Dus nadat ik mijn eerste opdracht had afgewerkt, heb ik gedurende 3 à 4 weken uitgebreid onderzoek gedaan over de *Room library*. Daarna ben ik begonnen met deze kennis om te zetten in code door dit te implementeren in de *Overtime* app. Ik heb een groot deel van de applicatie helemaal moeten herschrijven omdat er een groot verschil is tussen *Realm* en *Room*. *Realm* is namelijk geschreven volgens het *NoSQL* principe wat in het kort wilt zeggen dat er voor *Realm* geen SQL-query's moeten geschreven worden. *Room* daarentegen wordt beschreven als een abstractie laag bovenop *SQLite* en daarvoor diende ik wel SQL-query's te gaan schrijven voor de database-interacties in de *Data Access Objects*. Dit heeft me wel wat tijd gekost omdat het al enige tijd was geleden dat ik nog query's had moeten schrijven. Zoals in de onderstaande screenshot duidelijk wordt is er een groot verschil tussen de twee om dezelfde functie te schrijven.



```

Realm: fun findFirstById(id: Int) = where().equalTo( fieldName: "id", id).findFirst()

Room: @Query( value: "SELECT * FROM entries WHERE id = :id AND deleted_at IS NULL")
suspend fun getEntryById(id: Int): Entry
  
```

Wat wel direct merkbaar was is dat *Room* betere ondersteuning biedt voor de ingebouwde functies van *Jetpack*. Daarnaast kon ik ook makkelijker gebruik maken van de *coroutines* functionaliteit van Kotlin. Dit is zeer positief om de app beter op een asynchrone manier te laten werken. Zo is er ook de mogelijkheid voor *cross-threading* van objecten en de makkelijke integratie met *LiveData*. Deze aspecten zorgden ervoor dat ik een grotere controle had over wat ik kon doen met de data van de app. Allemaal voordelen aan *Room* dus die ervoor zorgen dat ik een moderne en beter functionerende versie van de *Overtime* app kon ontwikkelen.

Enkele moeilijkheden waren er natuurlijk ook: omdat *Realm* heel uitgebreid is, zijn er verschillende functionaliteiten die ik nu niet meer kon gebruiken, waarvoor ik dan andere oplossingen heb moeten zoeken. Zo is er in *Realm* een functie "*createOrUpdateObjectFromJson*" waarmee ik de JSON-call naar de onlinedatabase in een JSON-object kon zetten. Op dit object kon ik dan verschillende opdrachten uitvoeren zodat ik een grotere controle had over de objecten die ik uiteindelijk in mijn lokale database ging bijhouden. Bij *Room* heb ik dit moeten oplossen door onmiddellijk een tijdelijk object aan te maken wanneer de JSON-call gebeurt en de data hierin te parsen. Dit tijdelijk object

moest van het type zijn dat bij de opgevraagde data hoorde. Daarna kon ik dit object doorgeven aan het bijhorend *Data Access Object* en daarin het object in de lokale database plaatsen. Het feit dat ik niet langer met JSON-data kon werken heeft voor enkele problemen gezorgd wanneer ik wou werken met lijsten van objecten. Deze problemen heb ik dan opgelost door een andere type object te maken dat de lijsten wel ondersteunde en dat tijdelijk als tussenstuk te gebruiken.

Naast het gebruik van *Room* heb ik tijdens deze opdracht ook de kans gekregen om meer te leren over het gebruik van *LiveData* en *Observer*-objecten. Hiermee heb ik ervoor gezorgd dat wanneer de data veranderd deze verandering wordt gecommuniceerd naar een *Observer*-object. Dankzij deze handeling kan de UI dan in het *Observer*-object worden aangepast. Dit alles volgt de "levenscyclus" van andere componenten van de app, dat wil concreet zeggen dat wanneer een bepaald scherm niet meer zichtbaar is er ook geen aanpassingen meer gebeuren aan de componenten van dat scherm. Zo wordt het veel makkelijker om geheugenlekken te voorkomen.

Ook heb ik enkele malen raad moeten vragen aan mijn stagementor en andere leden van het Android-team in verband met code die door hen al was geschreven en helemaal gericht was op het gebruik van *Realm* waarin ik dan ook enkele aanpassingen heb moeten doen.

Ik heb 7 weken de tijd gehad om het onderzoek te doen en de applicatie te herwerken. Uiteindelijk heb ik de applicatie grotendeels kunnen afwerken maar ik heb mijn bevindingen na het onderzoek omwille van tijdsnood niet meer kunnen presenteren aan het Android-team. Ik hoop dit in de toekomst nog te kunnen afwerken.