Realm: the real-time database by MongoDB

Luigi D’Onofrio - 1710371

Overview on MongoDB

- It is a cross-platform, document-oriented database
- It provides high performances, high availability and easy scalability
- It works on the concept of database, collections and documents
- MongoDB stores data in flexible JSON-like documents

```
{
   _id: ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100,
   comments: [
      {
         user:'user1',
         message: 'My first comment',
         dateCreated: new Date(2011,1,20,2,15),
         like: 0
      },
      {
         user:'user2',
         message: 'My second comments',
         dateCreated: new Date(2011,1,25,7,45),
         like: 5
      }
   ]
}
```

Advantages of MongoDB

- **Schema-less:** which means that it does not enforce a schema, fields can vary from document to document and data structure can be changed over time

- **Replication**: the process of synchronizing data across multiple servers, it is important for data availability and also allows the users to recover from hardware failure and service interruptions

- **Sharding**: the process of storing data records across multiple machines, it is an approach that allow to meeting the demands of data growth; As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput. Sharding solves the problem with horizontal scaling.

What is Realm?

Realm is an embedded, object-oriented database that lets you build real-time, offline-first applications.
Its SDKs also provide access to Atlas App Services, a secure backend that can sync data between devices, authenticate and manage users and also run serverless JavaScript functions.
Realm solves many of mobile/web challenges that the developers can found in their work. These challenges include network reliability, local storage, and keeping reactive UI.

1. **Network reliability**: it is offline-first, you always read and write to the local database, not over the network
2. **Local storage**: access objects using the native query language for each platform
3. **Reactive UI**: Live objects always reflect the latest data stored in Realm Database

ACID compliance

Realm Database guarantees that transactions are ACID compliant.
This means that all committed write operations are guaranteed to be valid and that clients don't see transient states in the event of a system crash.

- **Atomicity**: groups operations in transactions and rolls back all operations in a transaction if any of them fail
- **Consistency**: avoids data corruption; if the result of any write operation is not valid, Realm cancels and rolls back the entire transaction
- **Isolation**: allows only one writer at a time
- **Durability**: writes to disk immediately when a transaction is committed.

Realm for Android app

Realm is a good alternative to SQLite database for Android application, but with much more powerful built-in tools.

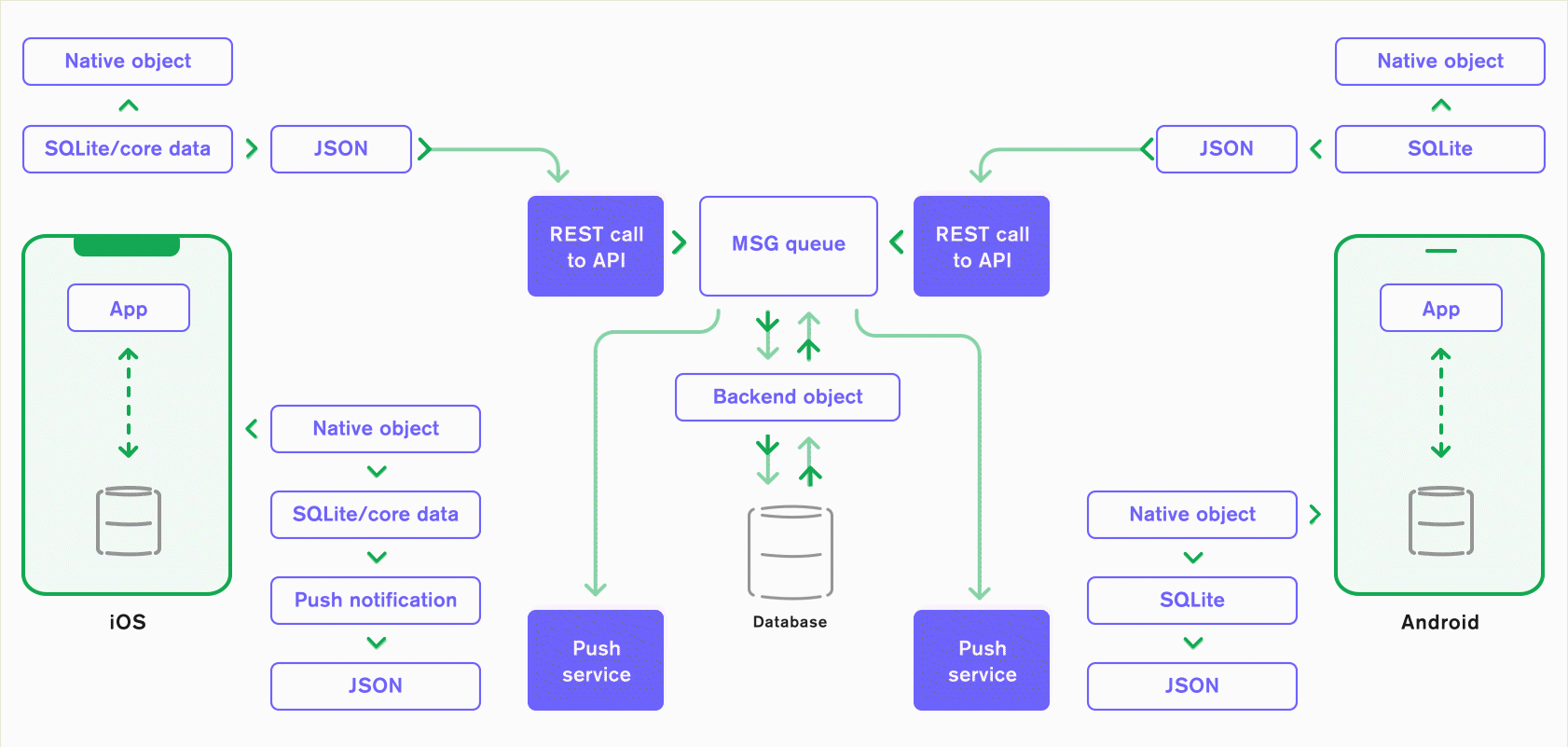
Authentication and Registration

Realm provides API for authenticating and registering users using any enabled authentication provider, among the most famous Google, Facebook, Apple or simple with email and password.

Synchronization

Atlas Device Sync automatically synchronizes data between client applications and an Atlas App Services backend. When a client device is online, Sync *asynchronously* synchronizes data in a background thread between the device and your backend App.

How MongoDB has simplified the complex architecture...



A key topic: Partition – part 1

MongoDB Realm Sync lets a "user" access their application data from multiple mobile devices, whether they're online or disconnected from the internet. The data for all users is stored in MongoDB Atlas. When a user is logged into a device and has a network connection, the data they care about is synchronized. When the device is offline, changes are stored locally and then synced when it's back online.

Why bother limiting what data is synced to a mobile app?

1. **Capacity**: we don’t want to waste limited resource on a mobile device to store data that the user has no interest in
2. **Security**: some users could not see some data in terms of privacy, and it is safe to not store it on their device

A key topic: Partition – part 2

There are different useful partitioning strategies to adopt. The most used:

- **Firehose**: this is a decision not to partition the data, when the volume of the data is not so high and there is nothing to maintain private
- **User**: each user has a unique ID, and each document contains an attribute that identifies the user that owns it, in this way only the data that is unique to the users is stored.
- **Team**: this strategy is used when you need to share data between a team of users
- **Channel**: with this strategy, a user is typically entitled to open/sync Realms from a choice of channels

How to integrate Realm Database in our app

The Object Model

As you continue to develop your application, you will need to modify your data model. While Development Mode is on, you can edit your Realm Object Model in your client code. Data Validation occurs when Development Mode is off, so Atlas App Services does not accept changes to your Realm Object Model while Development Mode is not on.

To work with Atlas Device Sync, the data model must have a primary key field called **_id**

```
open class Note(
    var noteName: String = "",
    var date: Date = Date(),
    var _partition: String = "Public",
    var owner: String = ""
):RealmObject() {

    @PrimaryKey
    var _id: ObjectId = ObjectId()
}
```

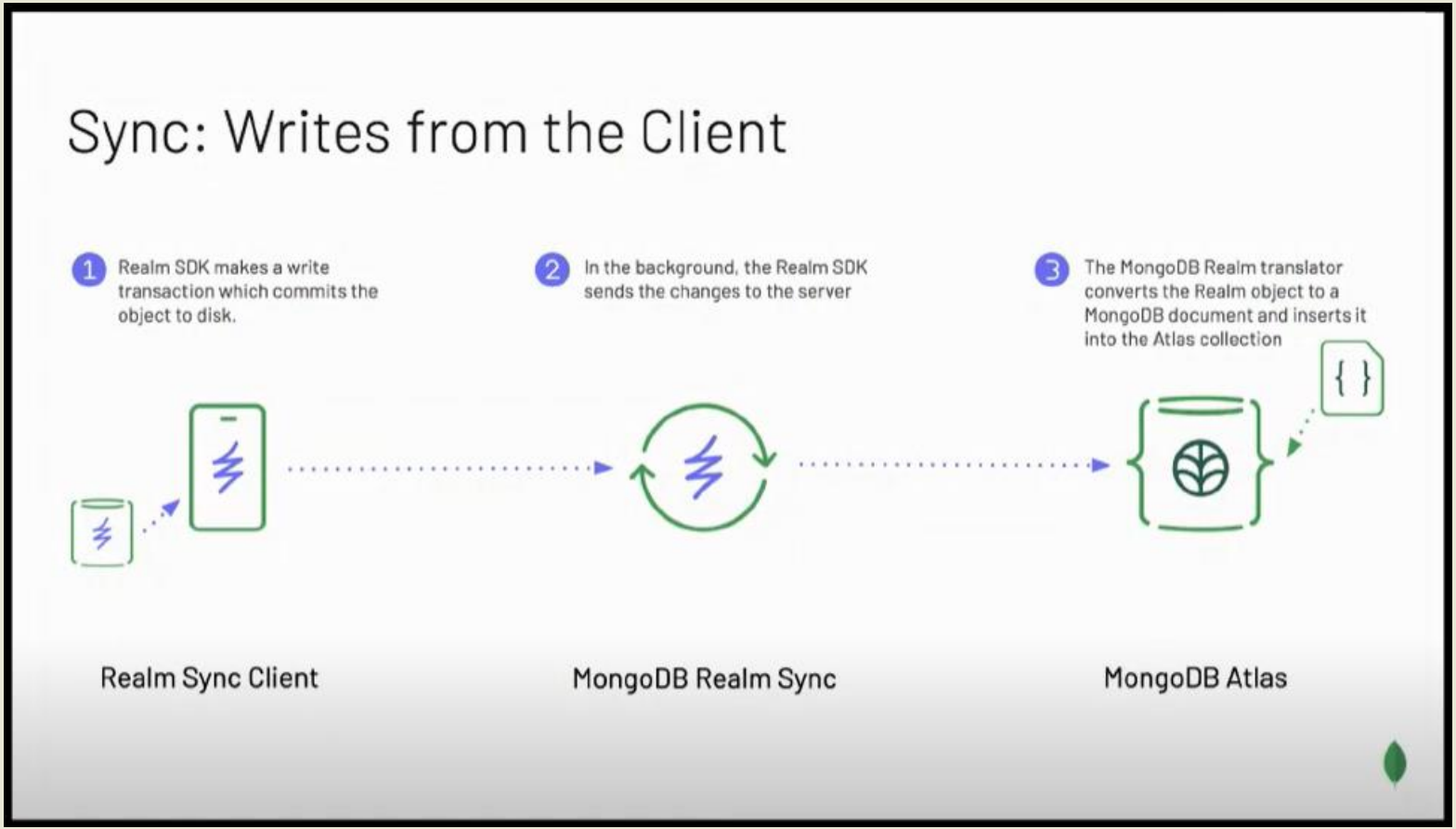
```
open class User(
    @PrimaryKey @RealmField("_id") var id: String = "",
    var _partition: String = "User",
    var name: String = "",
    var surname: String = "",
    var email: String = ""
): RealmObject() {

    override fun toString(): String {
        return "User [name=$name, surname=$surname, email=$email]"
    }
}
```

Open a Realm Sync

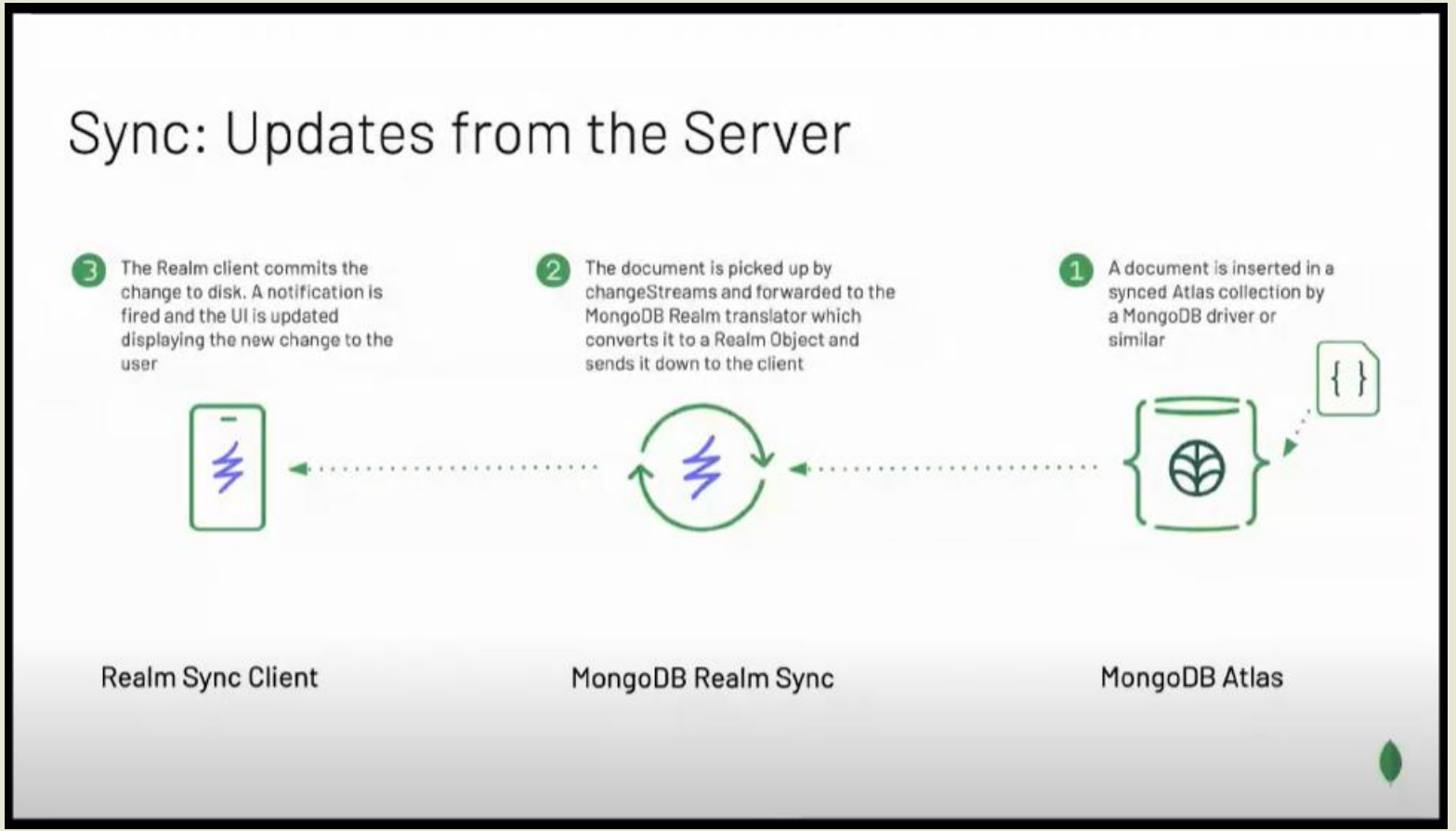
To open a synced realm, pass a user, a partition to SyncConfiguration.Builder()
Then we have to open an instance of the realm.

```
val config = SyncConfiguration.Builder(user, "Public")
                .build()

            Realm.getInstanceAsync(config, object : Realm.Callback() {
                override fun onSuccess(realm: Realm) {
                    this@HomeFragment.realm = realm
                    setUpRecyclerView(realm)
                }

            })

...
...

adapter = NoteAdapter(realm.where<Note>().sort("date").findAll())//.equalTo("owner", user!!.id)
```

Synchronization – part 1



Synchronization – part 2



Overview of the application – Note App