Couchbase

How to Choose a Database for Your Mobile Apps

Evaluating your mobile database – A checklist

Successful mobile apps rely heavily on their database provider.

How does your mobile database stack up when put to the test?

- Does it have support for the right platforms?
- Can you control your database sync?
- What sort of querying options does your local database support?
- Is data secure at rest and in motion?
- Do you deploy on-prem or in the cloud?
- Does the sync technology support multiple topologies?
- Do you worry about adapting to changing data schemas?
- How does your database handle pesky conflicts?
- Should you build or should you buy?

How to choose a database for your mobile apps

The choice of database plays a key role in building successful mobile apps. Data synchronization, local data storage and querying capabilities, and end-to-end security are key elements of a data platform.

Today's consumers are highly reliant on their mobile applications, with large portions of modern business relying on (or will rely on) the ability to interact with their users via mobile. If apps don't work, users won't use them – it's that simple.

To avoid reliance on a network, providers of databases and cloud services have started to add synchronization and offline capabilities to their mobile offerings. If apps require a connection to work, then the end-user experience will be sluggish and unpredictable.

Solutions like Couchbase Mobile, MongoDB Mobile, Amazon's AWS AppSync, and Google's Cloud Firestore offer the all-important sync that enables apps to work both online and offline. Other database providers such as SQLite, Room, and Core Data support local storage but do not provide sync support.

With so many offerings available, how does a mobile developer select the right technology? The following key criteria are essential when evaluating mobile solutions: multi-platform support, local data storage capabilities, sync capabilities with conflict resolution, ease of development, security, agile data modeling, flexible deployment, and topology options.

Support for the right platforms

What client platforms are supported? Do you need to go beyond iOS and Android? Are you looking to support platforms that aren't traditionally considered mobile, such as embedded systems and IoT devices? Are you looking to support Windows and Mac desktops and laptops as well? What about cross-platform technology?

Many of today's applications start on mobile, then add a native desktop version.

It is important to evaluate database and cloud options based on the platform support that you need not only today but also in the future.

Secure at rest and in motion

When you're using synchronized and decentralized storage it is important to access, transmit, and store data securely. To cover this completely, you need to address authentication, data-at-rest, data-in-motion, and read/write access control.

Authentication should be flexible and allow for the use of common public and custom authentication providers. Support for anonymous access is also important for many apps. For data at rest on the server and client, you'll want support for both file system encryption and data-level encryption. For data in motion, communication should be over a secure channel like SSL or TLS. For data read/write access, the database should include granular user and role-based access control (RBAC).

Flexible data models

Data modeling flexibility will dictate whether you can articulate the model requirements for your apps in an efficient and appropriate way. Model flexibility is especially important in mobile because today's mobile apps evolve at a very fast pace, and it will dictate whether you can easily adapt your model as your requirements change in the future.

A new release of a mobile app with an updated data schema will require expensive database schema migrations to be performed on app launch, adding to app startup costs. As app developers, you do not have control over when a new version of your app gets adopted which means you would potentially have users migrating from a very old version of schema to the latest, exacerbating the data migration issue.

Relational databases are still a good choice if an app requires strong data consistency or its data is highly relational. But when these requirements can be relaxed, NoSQL databases offer much greater flexibility.

Sync with the right partitions

Configurable sync topology support is needed to allow you to meet your partition requirements. In other words, you need the ability to configure the system to allow certain parts to operate offline. A star network is the most common topology. In a star topology, each device is connected to a central hub using a point-to-point connection that allows the devices to operate offline. Other common topologies such as tree and mesh allow different parts of the system (in addition to the devices) to operate offline.

You may also want support for cloudless topologies that allow devices to communicate peer to peer and directly sync data among themselves. Peer-to-peer synchronization is a powerful addition or alternative to client/server synchronization. It allows apps to connect and exchange data directly without going through a central server in the cloud. As a result, apps can continue to work and share data regardless of internet availability.

A point-of-sale (POS) system is a good example of a tree topology. POS systems require that a brick-and-mortar store continue to operate if it becomes disconnected from the rest of the system. In this configuration, POS devices would sync with a store-level database, which would then sync with a global system. So, stores can continue to operate and sync data with their POS devices regardless of connectivity to the global system.

Pesky conflicts

For mobile platforms or any other platform that utilizes decentralized data writes, the same data can be simultaneously modified on multiple devices, creating a conflict. The system needs to support a mechanism for resolving those conflicts.

Conflict handling will differ for each system. Couchbase Mobile, for example, uses revision trees with a default resolution rule of “most active branch wins.” This is the same approach taken by revision control systems such as Git and much different than clock-based systems that take a “most recent change wins” approach. Clock-based resolution systems are problematic due to the issues around clock differences across devices.

Sync at the right times

In addition to being able to resolve conflicts, it is important to have the ability to control how the system syncs, which includes replication strategy, replication events, conditional replication, and replication filtering. For replication strategy, look for support for streaming, polling, one time, continuous, and push. The granularity of sync has a direct impact on network bandwidth usage, so having a solution that is smart about identifying what subset of data has changed and only syncing the deltas is an important consideration in mobile deployments where data plans come at a premium and network bandwidth is limited.

You should also have the ability to use a combination of these strategies. For replication events, you may need to know the overall replication status as well as the replication status of individual documents. For conditional replication, you may need to replicate data only under certain conditions, such as when the device is on Wi-Fi or when it has sufficient battery power. For replication filtering, you should have the ability to replicate some data but not other data. The filtering could be fine-grained and could be based on the content of the data itself.

Flexible deployment

How you deploy the database is a very important decision and one that should not be constrained by requirements to use a particular hosting provider or platform. Avoid vendor lock-in so that your ability to grow or migrate projects in the future is not limited by functionality or cost issues. The ultimate solution will allow you to host your database on any public or private cloud as well as on-premises in your own datacenter. Ideally, you would not need any additional third-party hosted services or additional software to build your complete mobile stack.

Other ways to ensure flexibility is to use modern containerization approaches that make it easy to manage deployments as needs change or grow. Using Kubernetes to help orchestrate your containerized environments makes it much easier for your operations team to keep things manageable.

Local data storage capabilities

When looking for offline storage capabilities, you will need to determine if your app is expected to be usable in completely standalone mode with extended periods of network disconnectivity or if you are looking for a temporary cache to handle short network disruptions. An extensive database query API with full-text search support and the ability to be asynchronously notified of database changes will allow mobile apps to support responsive and highly reactive workflows locally. An intuitive, easy-to-use API will reduce ramp-up costs and reduce development and integration efforts.

Should you build or should you buy?

When looking to add sync to your apps, you will need to determine if you should build a solution or get it from a provider. Building sync correctly is notoriously difficult and expensive, as it must deal with all of the complexities of distributed computing. For most apps, you will be better off leaving data synchronization to a specialized stack and focusing on your app features. The key is choosing a solution that is flexible. If you go down the build path, be ready to expend a significant portion of your time and resources on building sync and supporting everything listed above.

When choosing a mobile sync and storage provider, taking full measure of the above criteria will be critical to building secure, flexible, and manageable mobile apps that always work – with or without an internet connection.

How the major players stack up

Capability	Couchbase Mobile	MongoDB	Google Cloud Firestore	AWS AppSync	SQLite, iOS Core Data, Android Room
Offline support					
Platform support					
Enterprise-level security					
Flexible data model					
Flexible topology support					
Peer-to-peer sync					
Delta synchronization					
Flexible deployment					

Why Couchbase Mobile?

Couchbase Mobile brings the power of NoSQL database to the edge. It includes Couchbase Lite, an embedded NoSQL database for mobile apps that exposes a SQL-based query API and the Sync Gateway, a synchronization gateway responsible for synchronizing data across clients and the cloud, for enforcing access control policies, authentication, authorization and data routing. Couchbase Mobile offers end-to-end Enterprise grade security.

Learn more at www.couchbase.com/mobile.

About Couchbase

Couchbase's mission is to be the database platform that enables a revolution in application innovation. To make this possible, Couchbase created an enterprise-class NoSQL database to help deliver ever-richer and ever more personalized customer and employee experiences. Built with the most powerful NoSQL technology, Couchbase was architected on top of an open source foundation for the massively interactive enterprise. Our geo-distributed database provides unmatched developer agility and manageability, as well as unparalleled performance at any scale, from any cloud to the edge.

Couchbase has become pervasive in our everyday lives; our customers include industry leaders Amadeus, AT&T, BD (Becton, Dickinson and Company), Carrefour, Cisco, Comcast, Disney, DreamWorks Animation, eBay, Marriott, Neiman Marcus, Tesco, Tommy Hilfiger, United, Verizon, Wells Fargo, as well as hundreds of other household names. For more information, visit www.couchbase.com.

3250 Olcott Street | Santa Clara, California 95054 | 1-650-417-7500 | www.couchbase.com