

Building Real-time Mobile Apps that Scale

Introduction

Mobile has impacted every aspect of our lives. Whether we are looking to schedule grocery deliveries, find our next apartment, or schedule a doctor appointment, we can confidently say “there’s an app for that”.

Conditioned by years of chart topping apps like TikTok, Instagram, and Spotify, mobile users expect to be able to collaborate in real-time without noticeable delays. Users also expect to feel connected at all times even if their internet connection is spotty – nothing’s more frustrating than seeing a gray screen with the dreaded “no internet connection”.

This means mobile developers need to build reactive, collaborative, always-on experiences into all of their mobile apps. But building real-time mobile apps is hard. It requires keeping a mobile database and the backend in sync at all times regardless of connectivity issues or data conflicts. Traditionally, developers have had to turn to one of two approaches:

1. **Use REST APIs** to connect the client-side mobile database to the backend. However, building a synchronization mechanism is complicated and requires thousands of lines of complex, non-differentiating code – reducing development

velocity, compromising user experience, and escalating costs.

2. Integrate **off-the-shelf synchronization solutions** that work between your mobile database and your backend to keep the two in sync. Off-the-shelf solutions give developers the ability to build an MVP app fast. However, the development time saved during MVP is spent later when developers need to build out lots of custom code to compensate for the limitations of overly simplistic sync engines – again leading to major bottlenecks when looking to scale down the road.

[MongoDB Atlas Device Sync](#) gives you a much better way. With industry-leading features like field-level permissions and dynamic querying based on user inputs, Device Sync provides developers with total control over what data is synced and when, enabling even the most complex use cases out-of-the-box.

In this white paper, we dig deeper into the challenges of building real-time mobile apps that scale today and how that’s transformed with Atlas Device Sync. We discuss use cases for Device Sync and provide real examples of how thousands of businesses are using it today.

The Trouble with Real-Time

Although a real-time experience is essential for every mobile application, building it isn’t easy. Application owners need to consider how to store data on device when connectivity drops, how to sync that data back to cloud upon reconnection, and how to handle any conflicts that might emerge if multiple users or devices were making changes at once.

To address these requirements, developers use a three part architecture: 1) a mobile database 2) a cloud database and 3) a synchronization mechanism that keeps data up-to-date across users, devices, and the backend.



Option 1: Build it yourself

Because it is easy to understand the concepts of design for real-time, building it yourself would seem to be the simple choice. You can maintain control over what data is synced between the device and the cloud, designing a solution specific to your apps needs.

However, as developers begin to build for real-time, what may have initially seemed simple to build yourself reveals itself to be significantly more complicated in practice.

The hidden pains of data synchronization

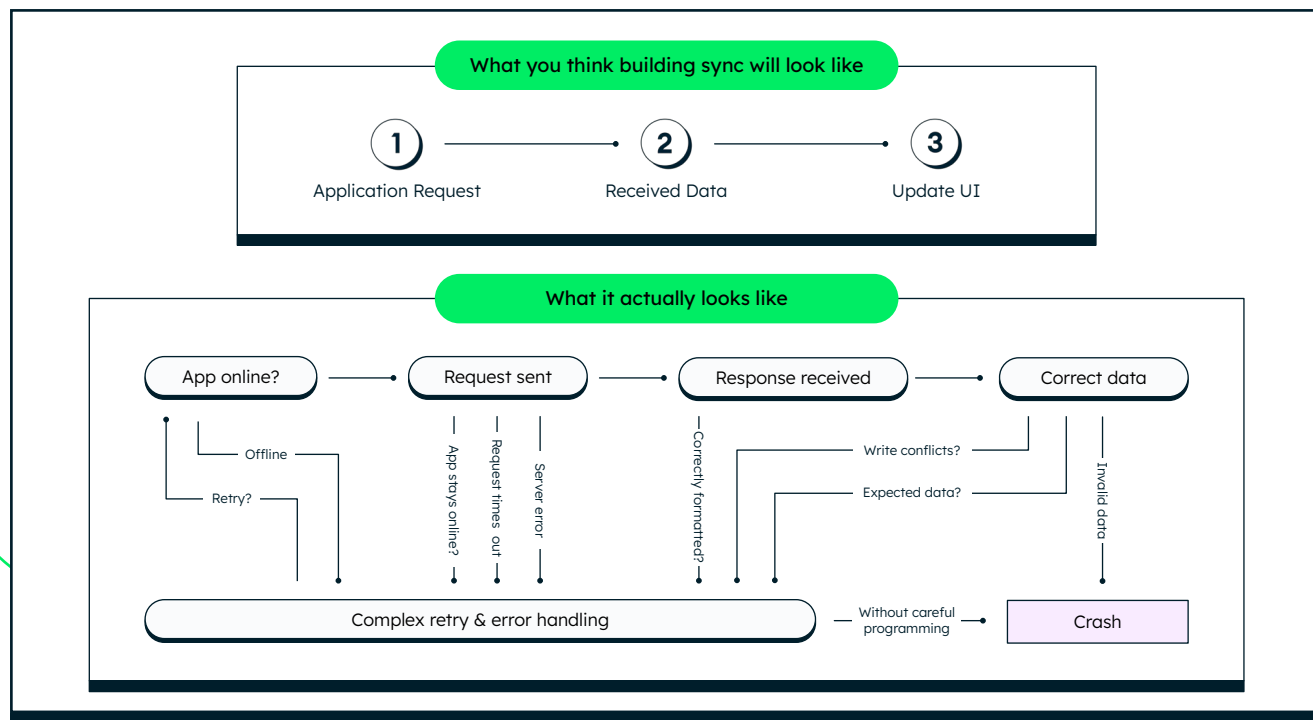
Many developers initially view data synchronization as three steps: 1) application request 2) received data and 3) update UI. What that process fails to account for is the host of technical challenges that come with building for mobile.

These include:

- **Supporting offline mode.** When data is requested, applications need to understand whether a network is available, and if not, whether the appropriate data is stored locally, leading to complex query, retry, and error handling logic.

- **Handling lost connections.** When data is written offline, it must be flagged or tagged as such. When the app reconnects, any tagged data is sent to the server. It's not enough to just save the changes locally. Even if data is written when an app is online, your logic needs to handle add the edge cases of network failure/retry or risk losing data.
- **Managing conflict resolution.** Imagine a scenario in which an offline user deletes data while another offline user updates it at the same time. It's difficult to determine the final state of this information. Who should win?
- **Balancing technical trade-offs.** As you build your data sync, you must consider the size of your app, how much space it consumes on the user's device, and its power consumption.

On top of these unique technical challenges are all the traditional concerns of building any piece of software: security, performance, stack expertise of current staff, and the ability to hire support engineers. What may have initially seemed simple to build yourself reveals itself to be significantly more complicated in practice.



Option 2: Integrate an off-the-shelf synchronization service

A better option is to use an existing service that has already solved all of the problems detailed above. Strong synchronization solutions enable handling intermittent network connections and conflicting changes to data, as well as offer comprehensive access controls and security features.

Many off-the-shelf services appear to satisfy these requirements on paper and make it easy to build to MVP. But fall short when scalability and data complexity become factors.

Hitting the limits of off-the-shelf solutions

Many synchronization solutions work well for quickly iterating to a minimum viable product. However, many apps quickly reach the limitations of the off-the-shelf solutions and developers revert back to move to more custom solutions to gain flexibility, control, and performance.

Common limitations include:

- **Query constraints.** The lack of aggregation support is common in off-the-shelf solutions and can often lead to the overfetching of documents – forcing developers to handle operations in their own code and slowing offline app performance.
- **“Last write wins”.** Most services default to the “last write wins” approach to conflict resolution which could lead to significant data loss if multiple users are writing to the same document while offline.
- **Permissioning.** Many apps handle sensitive data like PII where it is important to only sync data based on a user’s role. This level of granularity and control is oftentimes missing all together from out-of-the-box solutions.
- **Cloud Vendor Lock-in.** Many services only work with one cloud vendor leading to lock-in issues down the road.

To handle these challenges, many developers combine off-the-shelf solutions with custom code and workarounds, complicating the architecture and impacting the long-term scalability and performance of the application.



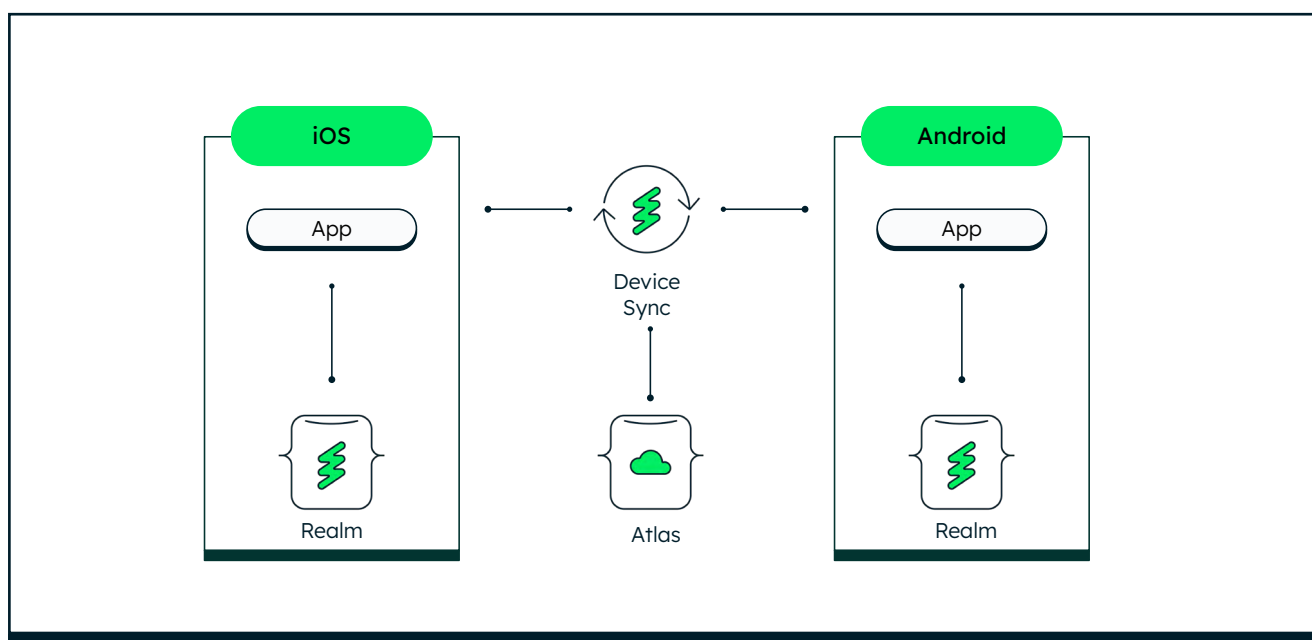
MongoDB Atlas Device Sync: A Better Approach

[MongoDB Atlas Device Sync](#) is the fastest and easiest way to build and maintain real-time, reactive mobile apps that scale. It combines the power of [Realm](#) – the same mobile database

underpinning the world's most popular applications – with the developer productivity, scale, and resilience of [MongoDB Atlas](#).

How does Atlas Device Sync work?

Atlas Device Sync is part of the MongoDB Atlas developer data platform. Its data synchronization service works alongside Realm to synchronize data bi-directionally, between Realm on the client and MongoDB Atlas in the cloud.

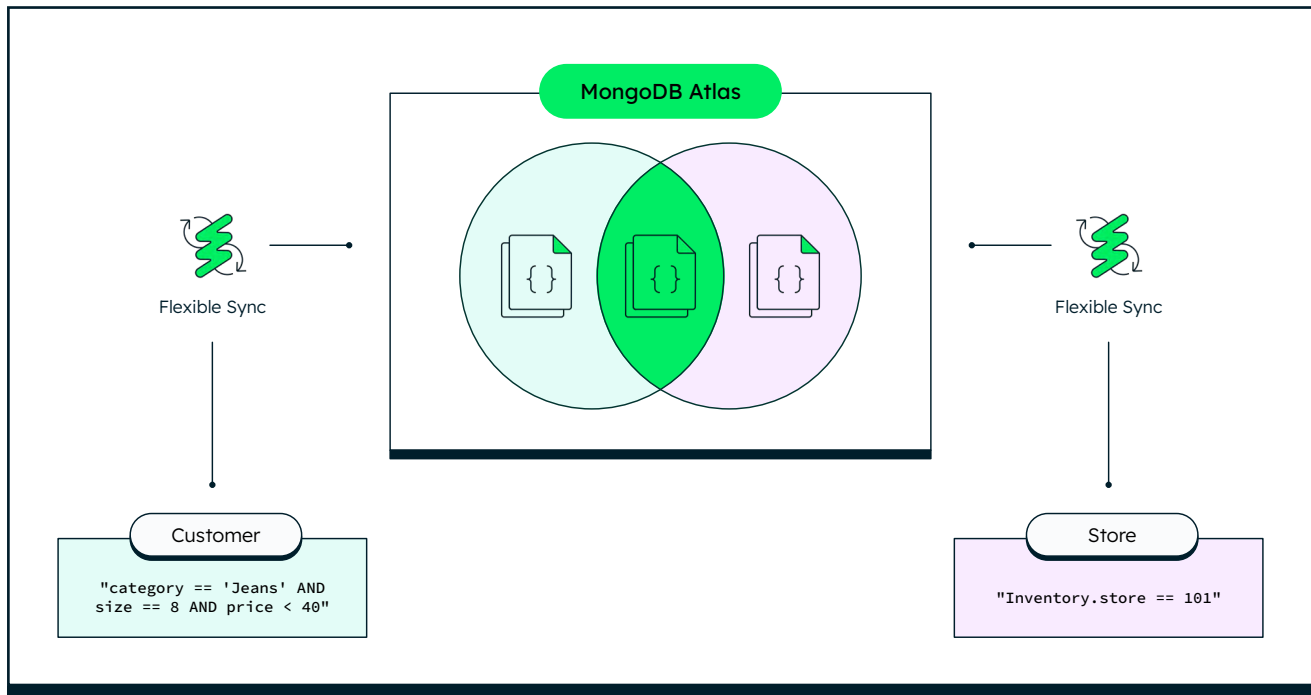


Dynamic querying

Atlas Device Sync lets you use language-native queries to define the data synced to user applications. This more closely mirrors how you are used to building applications today – using GET requests with query parameters – making it easy to learn and fast to build to MVP.

Device Sync avoids the typical query limitations of other solutions by supporting dynamic, overlapping queries based on user inputs.

Picture a retail app that allows users to search available inventory. As users define inputs – show all jeans that are size 8 and less than \$40 – the query parameters can be combined with logical ANDs and ORs to produce increasingly complex queries, and narrow down the search result even further. In the same application, employees can quickly limit inventory results to only their store's stock, pulling from the same set of documents as the customer, without worrying about overlap.



Advanced conflict resolution

Beyond the 'last write wins' approach that is offered in many out-of-the-box solutions, Device Sync offers operational transformation – a set of rules that guarantee strong eventual consistency.

Strong eventual consistency means all clients' versions will eventually converge to identical states. This will be true even if changes were made in a different order like in situations where multiple devices are writing while offline.

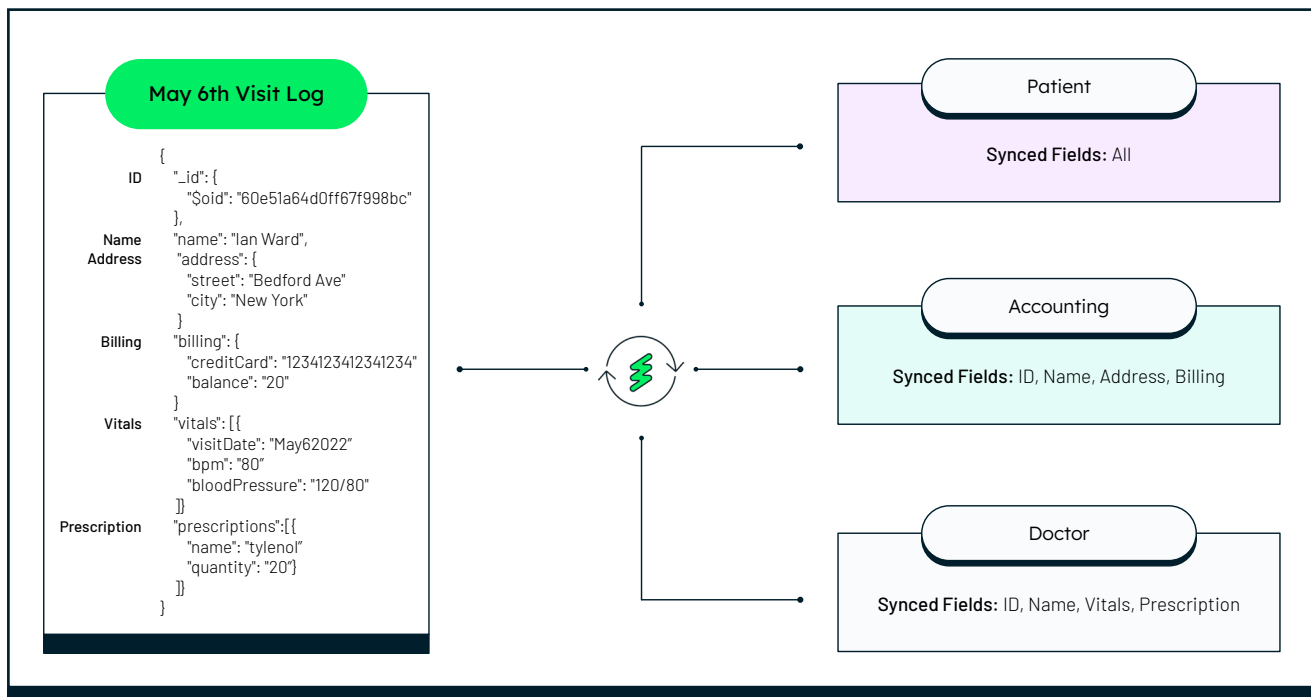
The conflict resolution engine follows four rules: deletes always win, the last update wins at the field-level, inserts in lists are ordered by time, and increment/decrements will be preserved.

Fine-grained permissioning

Whether it's a company's internal application or an app on the App Store, permissions are required

in almost every application. That's why Atlas Device Sync enables field-level permissioning when syncing data – meaning synced documents can be limited based on a user's role.

Consider how an emergency room team would use their hospital's application. A resident should only be able to access her patients' charts while her fellow needs to be able to see the entire care team's charts, and an administrator might be able to see all patients staying in the hospital, but not view any fields pertaining to their medical records. In Flexible Sync, a user's role will be combined with the client-side query to determine the appropriate result set. For example, when the resident above filters to view all patient charts the permission system will automatically limit the results to only her patients.



A complete developer data platform

Atlas Device Sync is built on top of MongoDB, the most popular and widely used modern database in the market. MongoDB has become so popular because engineering teams can build and ship

applications faster than other data platforms. You can get started with both MongoDB Atlas and Atlas Device Sync in minutes on a fully managed service that handles operations for you – on any cloud you choose.

Atlas Device Sync in Action

FloBiz is an Indian startup on a mission to help millions of SMBs move away from pen and paper and take advantage of digitization. The startup's flagship offering –myBillBook – helps over a million monthly users digitize their invoicing, streamline business accounting and automate workflows of their enterprises and runs entirely on MongoDB Atlas with the help of Device Sync and Realm.

Atlas Device Sync handles the difficult job of keeping the mobile, desktop, and web apps in sync. This means even if multiple users were using the same account, going offline and online, there would be no issues, duplications or lost data.

The large multi-national convenience store **7-Eleven** uses MongoDB Atlas Device Sync and Realm for an end-to-end inventory solution. The app provides a real-time and exact view of when sales and deliveries arrive. Currently, 8,500 stores use this app with data staying synced across 20,000 devices in real-time.

Cue Health is a healthcare technology company pioneering the digital transformation of

personalized health information, beginning with diagnostics. Cue Health's offering combines breakthrough science with connected software solutions that makes it easy for healthcare providers, enterprises, and individuals to manage real-time and actionable health information, offering easy access to lab-quality diagnostics anywhere, anytime in a device that fits in the palm of the hand. The company chose MongoDB Atlas, Search and Device Sync to power its Cue Health App.

A large French **fashion brand** is using Atlas Device Sync to power its mobile in-store checkout experience. Prior to Atlas Device Sync, the portable POS devices were constantly crashing and losing transaction data. Naturally, employees began reverting to the traditional checkout process.

With Atlas Device Sync and Realm, the POS systems have stopped crashing and reliably sync back to the cloud in real-time, so the brand's connected systems have the most up-to-date data to work with.




Getting Started with Atlas Device Sync

Atlas Device Sync is self-service and easy to get started with. If you don't already have one, sign up for a free [MongoDB Atlas](#) account.

Atlas Device Sync is available with all Atlas clusters – including free clusters – so you can evaluate it at no cost. Our Getting Started tutorial steps you through the process. Device Sync documentation provides and complete references on how to configure, manage, and deploy Sync in your application, along with performance recommendations. The MongoDB Developer Hub and MongoDB YouTube channel provide a wealth of articles and tutorials for beginners through to expert users.

Our professional services team can also support you at any stage of your application lifecycle. They can partner with you throughout a project to implement sophisticated solutions, including Atlas Device Sync and other components of the MongoDB Atlas developer data platform, as well as help derive longer-term strategic initiatives.

Whether you are building a new application, extending an existing MongoDB workload, or looking to simplify your mobile architecture, Atlas Device Sync makes it easy to get started, and makes your user experiences more engaging and delightful.

Try [Atlas Device Sync](#) on a free cluster today and see for yourself. 

Safe Harbor

The development, release, and timing of any features or functionality described for our products remains at our sole discretion. This information is merely intended to outline our general product direction, and it should not be relied on in making a purchasing decision, nor is this a commitment, promise, or legal obligation to deliver any material, code, or functionality.