The Offline-First Approach to Mobile App Development

Beyond Caching to a Full Data Sync Platform

TABLE OF CONTENTS

The Offline-First Approach to Mobile App Development

Beyond Caching to a Realtime Data Sync Platform

For many enterprise apps, lost connectivity can mean lost productivity, lost opportunity, or lost business. When a network connection drops, end users can't complete their task, or worse, they risk losing business-critical data. The vast majority of mobile apps depend on a network connection to provide even the most basic user experience, and when the network is unavailable, they are nonfunctional. This all-too-common experience greatly impacts user engagement for any app, which can significantly affect the business. Brands lose out to competing apps that are network-resilient. Organizations suffer as employees struggle with enterprise purposed apps in low connectivity environments.

Why are so many apps designed without offline in mind? Whether your organization is building apps for customers, enterprise productivity, or both, a poor offline experience will negatively impact your business.

In this white paper, we'll discuss the advantages of an offline-first development strategy for both developers and engineering managers. We'll look at the specific challenges of developing robust offline functionality in mobile apps and common technology solutions. You'll walk away with a better understanding of how to deliver a great offline user experience in your mobile apps.

"Offline support will be a crucial consideration for nearly every future modern application."
--Forrester, "The Offline Mobile Challenge," September 2014

The Challenges of Mobility

In recent years, mobile has surpassed desktop to become the world's dominant computing platform. This has been fueled in part by an explosion of apps that are designed to enable mobility. Mobile apps of all kinds are now central to our daily lives. We depend on them to fulfill the same functions as on a desktop, plus give us countless mobile-only experiences. Businesses depend on mobile apps to engage customers, increase workforce productivity, and establish new business models in a fast evolving marketplace. Mobility has become a way of life.

Underlying the promise of mobility are two fundamental user assumptions: 1) mobile apps will work wherever we are, and 2) mobile networks will be always available wherever we expect coverage. The former is possible with an offline-first approach, even though the latter is still unrealistic today.

In some environments, such as on an airplane or in a remote geographic area, we understand that network connectivity is not possible. However, we still expect a solid app experience, with a fully rendered UI, access to data, and as much functionality as possible offline. For example, we want to be able to read and write emails, or update calendars and contacts with confidence that our changes will hit the server automatically once we're back online.

Mobile networks have come a long way, but the reality is that they are still notoriously inconsistent. Even in the heart of a tech-savvy urban environment, network coverage can be spotty, and service can be congested, slow, or subject to intermittent fluctuation. Applications that rely on cloud-based services, data transfer, network speed, or various online processes are at risk for providing a poor user experience in the face of disrupted connectivity.

Network failures happen in every mobile app, not just those designed for unusual environments. Depending on the app, a degraded offline experience can drive away customers, disrupt employee productivity, or lose business-critical data. Bottom line: companies cannot afford to ignore offline usage.

From Mobile-First to Offline-First

Offline-first is a growing trend in mobile app development that considers offline use cases to be core business requirements. Like mobile-first apps, offline-first apps are purposefully designed to meet user needs in context. In these apps, network-resilient features are not simply bolted onto an existing app design, or handled as an afterthought. Instead, they are baked into the fundamental app architecture and user experience.

An offline-first app is based on an architecture and tech stack that enable it to handle data locally on device, and later sync that data to a server or the cloud when a connection becomes available. This allows the user to continue working—regardless of connectivity—and preserve his or her intended updates when back online. Moreover, if data is stored locally, the app makes fewer calls to the network and is thus less affected by network latency, producing a faster, more responsive user experience. Fewer network requests also means less drain on the device battery, a key issue for some users.

A well-designed offline-first app provides a seamless user flow between online and offline states. With data stored on device, the app can display the UI at all times, allowing the user to freely navigate throughout the app and view all data available at that moment. Although the data may not be entirely up to date when offline, there's always some representation of the data on device. If the app enables data entry, the user can still input data offline. Ideally, the user is never blocked from interacting with the app naturally while offline such that when a connection happens, syncing feels almost like a background process.

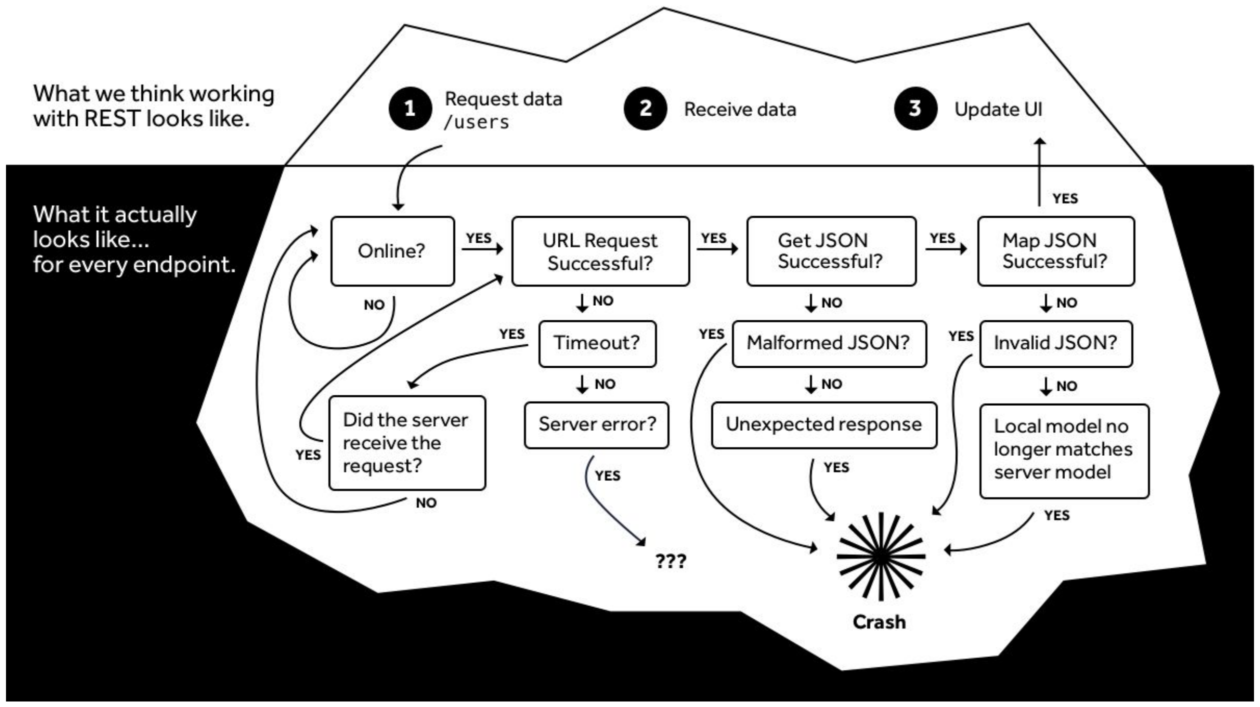
Why is Offline-First Development So Difficult?

Building a distributed system is one of the toughest challenges in computer science. An offline-first app is inherently a distributed system—data is stored on device and also on the server. Because data can change independently in either location, conflicts arise between versions of the data sets on device and on the server. This may seem like a simple problem, but conflict resolution can be astonishingly difficult to solve in app development.

When faced with the daunting complexity of resolving data conflict, developers tend to make a lot of compromises. Some will oversimplify their approach and reject changes coming from the device or focus only on handling errors. This usually means that the user will pay the price with a frustrating experience or unintended results. Other developers will attempt to build their own offline-first system from the ground up—with variable success.

"Offline support is the mobile app feature continually underscoped by developers and over-simplified by stakeholders."
--Forrester, "The Offline Mobile Challenge," September 2014

Both developers and engineering managers routinely underestimate the impact of offline-first development. A significant amount of developer time can be wasted trying to solve their app's offline use cases, and then build a solid offline-first system that can elegantly handle data flows, conflict resolution, error scenarios, and such. This slows down development and diverts engineering attention away from other product features.



Sending and receiving data back is just the tip of the iceberg. Networking requests can fail for a number of reasons out of your control. - "Best Practices and Pain Points in Mobile Networking", Marin Todorov, Realm.io

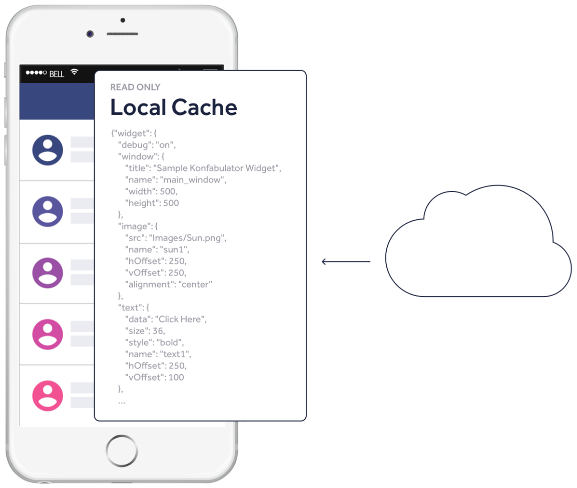
Not only will it take longer to build the app, but developers also end up with bloated code that requires even more time to debug, fine-tune, and, maintain and improve over time. Any developer will say "the best code is no code"—the less they have to write means fewer bugs, better performance, and faster delivery.

Common Approaches to Offline-First

One of the fundamental differences of an offline-first architecture is that the app always assumes data to be local. How that data is stored, accessed, and synced can vary. The following are three development approaches that range from simple to robust.

1. Caching

The most common approach is to simply cache data locally. Caching stores data for read access. This gives users access to data on their device, renders the UI, and provides a predictable user experience based on a snapshot in time. Caching can provide a smooth offline experience and faster performance—depending on the app, this may be enough to fulfill offline-first objectives.

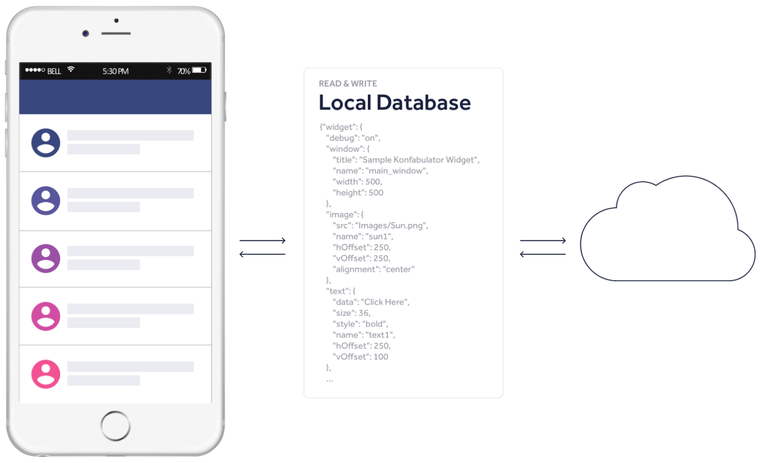


```
{"widget": {
  "debug": "on",
  "window": {
    "title": "Sample Konfabulator Widget",
    "name": "main_window",
    "width": 500,
    "height": 500
  },
  "image": {
    "src": "Images/Sun.png",
    "name": "sun1",
    "hOffset": 250,
    "vOffset": 250,
    "alignment": "center"
  },
  "text": {
    "data": "Click Here",
    "size": 36,
    "style": "bold",
    "name": "text1",
    "hOffset": 250,
    "vOffset": 100
  },
  ...
```

However, many apps are not simply read-only, but also enable data entry and manipulation. The major downside of caching for the user is that he or she is not able to update cached data, and is working in a limited capacity offline. A downside for the developer is that many caching solutions do not support queries, event handling, or server-side business logic, which requires the developer to write their own. Another common downside to caching is that it's difficult to know when a cache is invalid and data should be obtained again. If a single bit in a large cached document is different, the whole cache may be thrown out due to the lack of fine-grained detail about the freshness of portions of that cache. It's also often impossible to tell if a cache is out of date without downloading its entire contents from scratch, leading to overuse of a device's radio and rapid battery drain.

2. Manual Replication

This custom coded approach takes a step beyond caching to give read and write ability to the mobile client. It allows users to manipulate data while offline by marking objects as changed. When a connection becomes available, all changed objects are sent in full by triggering a push and pull operation that pulls the changed objects from the server and pushes the changed objects from the device.



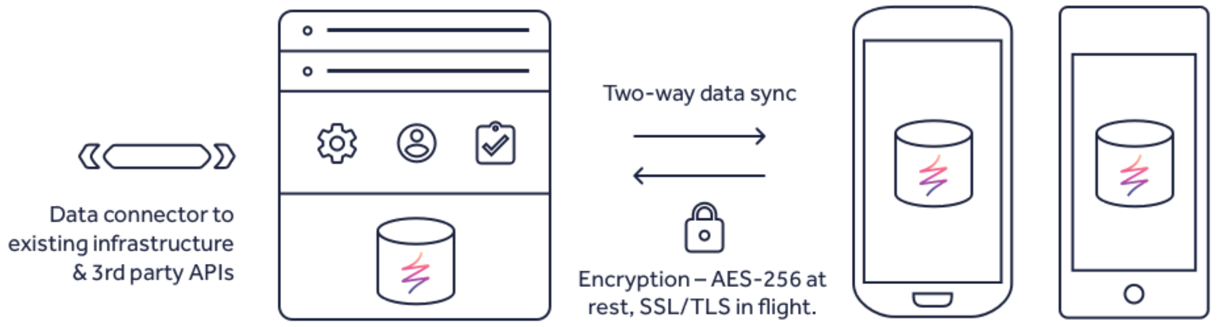
```
{"widget": {
  "debug": "on",
  "window": {
    "title": "Sample Konfabulator Widget",
    "name": "main_window",
    "width": 500,
    "height": 500
  },
  "image": {
    "src": "Images/Sun.png",
    "name": "sun1",
    "hOffset": 250,
    "vOffset": 250,
    "alignment": "center"
  },
  "text": {
    "data": "Click Here",
    "size": 36,
    "style": "bold",
    "name": "text1",
    "hOffset": 250,
    "vOffset": 100
  },
  ...
```

However, the method is unpredictable and can fail for any number of reasons stemming from the app, server, mobile network, or other points along the way. Because data is stored in JSON, ORM (serialization/deserialization into native objects) is required, which adds latency and the likelihood of conversion failures. More importantly, because the push and pull operations are independent, they can include changes to the same object. This forces the developer to worry about conflict resolution and its impact on app performance and user experience. In essence, this creates the same time-consuming headaches for developers as discussed above.

3. Realtime Data Sync by Platform

In this model, data sync is not manual, but automatic, sending changes in realtime. A highly efficient data synchronization protocol passes only the marginal changes in compressed binary format between device and server-side layer. Further, rather than sending complete objects, devoid of what specifically changed, this model focuses on synchronizing the specific operations along with the data. This additional information captures exactly what the user intended, allowing the system to automatically resolve conflicts, leading to predictable synchronization without manual intervention that impedes performance.

The Realm Mobile Platform makes this approach feasible by using a platform architecture that handles data synchronization between device and server. Data is stored in a transaction-capable database on device and a lightweight platform layer sits on the server. When the user updates data on the device, the local database and the server-side layer work as one to handle complexities of network state and resolve conflicts in realtime before writing the updates to the server-side database. Conversely, the platform manages data pushes from server to client, as well as broker messaging and collaboration functionality. The platform can also take care of such things as user presence, state events, server errors, and JSON decoding.

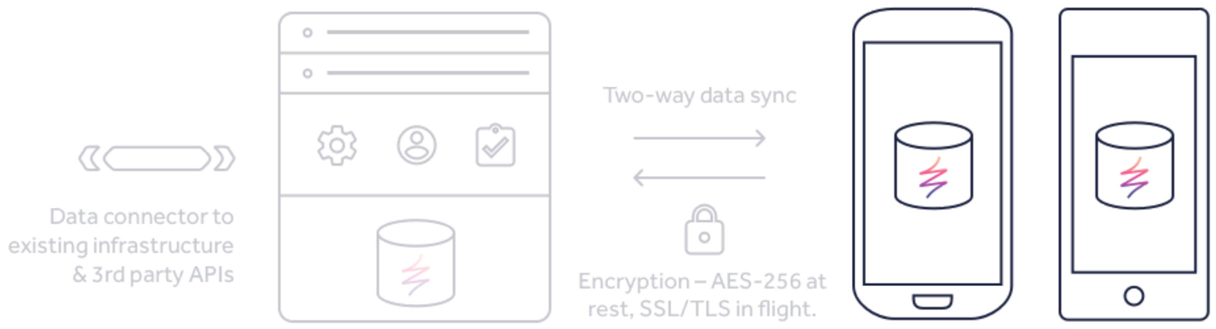

Data connector to existing infrastructure & 3rd party APIs
Two-way data sync
Encryption – AES-256 at rest, SSL/TLS in flight.

Four Ways a Platform Solution Solves Offline-First

The platform approach abstracts away the most challenging aspects of offline-first by providing a comprehensive solution that can be dropped in to any mobile app stack. By using a platform, developers no longer have to write lengthy, complex networking logic to sync data in realtime, resolve conflicts, or handle failures. The platform does all the heavy lifting for developers, while providing a responsive online/offline experience for users.

Four key components make a data sync platform especially powerful for app development.

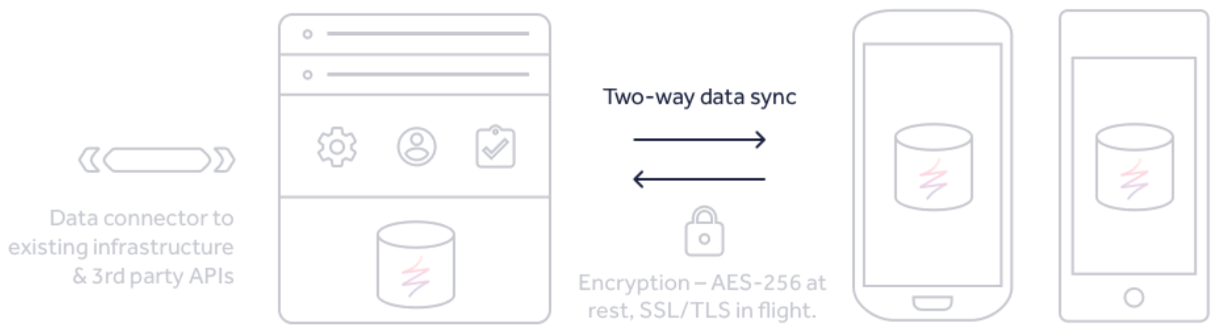
1. Robust Device-side Database


Data connector to existing infrastructure & 3rd party APIs
Two-way data sync
Encryption – AES-256 at rest, SSL/TLS in flight.

Going beyond a simple cache, a full database on device enables a much higher level of functionality, such as queries, change notifications, transactional changes, sorting, data mapping, data protections, safe concurrent access and other tasks. This can unlock new opportunities for enhancing the app's design, features, user experience, and even competitiveness. For example, if one retail app caches limited product data while another stores a full set of product data locally, then the app with the better user experience is likely to lead to stronger customer satisfaction, higher engagement, and may even generate more sales. If security is a concern, sensitive data can be encrypted or flushed after each user session, while the platform maintains a unified way of managing all data types.

Nowadays, forward-thinking developers are taking an endpoint computing approach, moving as much of their app's computation from the server to the mobile device. This not only helps to address network latency, but it also reduces the server-side burden and cost. This scenario is only possible if the app has data on the endpoint, which means it needs a reliable database on the device.

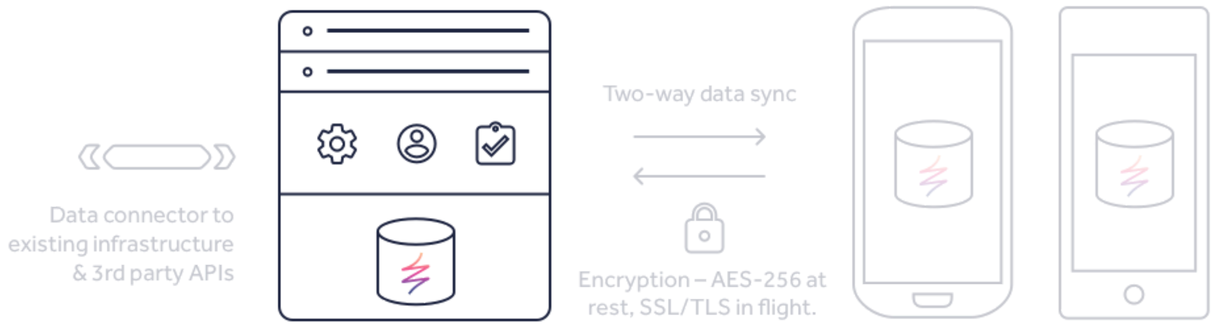
2. Realtime Sync


Data connector to existing infrastructure & 3rd party APIs
Two-way data sync
Encryption – AES-256 at rest, SSL/TLS in flight.

Very few apps today provide a realtime sync experience due to the extra steps they need to take to move data back and forth between the front-end and a traditional data store. All mobile apps are built around objects, which can be defined for the specific use cases of that app. But most databases do not use objects—they are more like very complicated, nested Excel tables with tables of data. Apps have to translate data back and forth between object form and table form, which slows them down and can introduce errors.

A new model supports a true realtime data sync experience for reactive apps by skipping the translation step and adding a platform layer to manage the logic. A comprehensive solution, such as the Realm Mobile Platform, provides an object-oriented database on device tied to an object-oriented server, with a data sync platform in between. As changes come in, live objects are updated in realtime on both the device and server-side data store, and the platform handles any conflicts, user authentication, or event triggers.

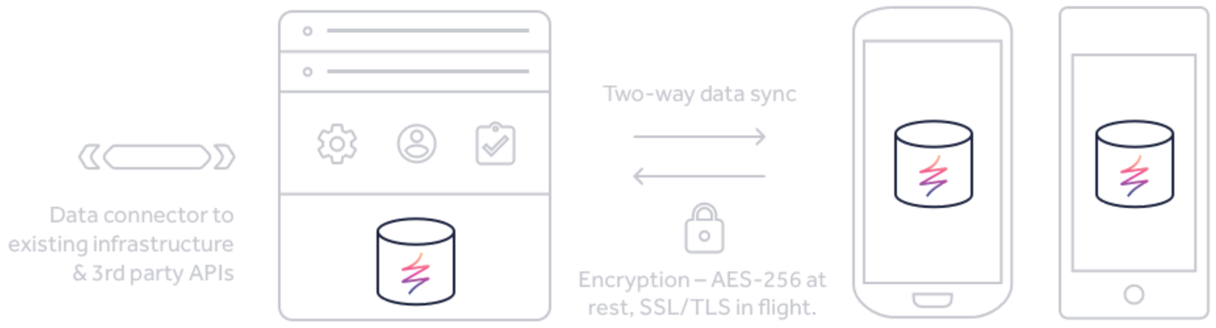
3. Conflict Resolution


Data connector to existing infrastructure & 3rd party APIs
Two-way data sync
Encryption – AES-256 at rest, SSL/TLS in flight.

One of the most difficult challenges that developers face with data sync use cases is around resolving data conflicts that happen when the user suddenly goes offline. For example, two users could be accessing and manipulating the same data at the same time, yet when they suddenly lose their network connection, they are no longer working on the same object. Which user's changes will get saved?

A data sync platform is responsible for automatically resolving data conflicts in realtime. The Realm Mobile Platform accomplishes this deterministically using pre-set rules and operational transformation—a research algorithm that was developed 20 years ago and popularized by Google in Google Docs. There's no need for developers to write reams of code to manually resolve conflicts with some form of version control or home-grown logic. The platform provides this complex functionality out of the box, with options to create custom conflict resolution rules.

4. Transaction Support


Data connector to existing infrastructure & 3rd party APIs
Two-way data sync
Encryption – AES-256 at rest, SSL/TLS in flight.

The most reliable and durable databases are ACID compliant and support transactions, both critical components to maintaining data integrity. Without transaction support, the developer has no way of ensuring that changes will be applied as intended. Transactions guarantee a reliable view of the data at all times.

With a data sync platform, the developer controls the whole stack and conflict resolution is done at the transaction level. A platform ensures consistency and safety of data, so that when changes are made offline due to a lost network connection, or the device runs out of battery, data is not lost.

Easy, Reliable Offline-First Implementation

To fulfill user demand for true mobility, offline-first has become a concern for all mobile app development. Yet most companies don't have the engineering resources, budget, or timelines to build their own solution from the ground up.

Modern app development integrates best-of-breed technical solutions to extend app functionality rather than code every aspect of the app in-house. Such solutions are built and maintained by organizations that are dedicated to excellence in their space, with experts focused solely on solving a particular problem. By integrating third party solutions, app developers are free to focus on what matters most—building business-critical features and experiences that help their company stay competitive in the market.

Realm Mobile Platform helps companies of all sizes build secure, network-resilient apps and gain better control of their stack. The platform powers worry-free, realtime data sync that does all the heavy lifting of offline-first functionality. It creates a reliable distributed system for data management in one integrated package, combining the popular Realm Mobile Database on device with Realm Object Server. In addition, Realm's flexible platform is easy to build on and helps developers keep app code lean, performant, and easier to debug and maintain.

Learn more about Realm Mobile Platform, or contact us for ideas on how to best to approach your app's offline-first use cases.

Realm