realm

Building Reactive Apps with Realm

Contents:

Introduction

For mobile application users, milliseconds matter. All users have a personal threshold of tolerance with respect to app speed. Slow performance, or the insufferable loading screen, immediately triggers our frustration. If our flow is repeatedly interrupted, we can easily lose our train of thought along with our momentum or productivity. We may even abandon the app altogether midstream. To deliver an engaging and immersive app experience, speed is essential. Apps must provide a lightning-fast UI and ensure resiliency against the latency inherent in mobile networks and many legacy systems—all in realtime.

As more and more businesses build customer-facing apps, competition for space on the mobile screen is more fierce than ever. Your app may have the best design and the most interesting features, but if its performance is sluggish or it stalls when offline, then your app is at great risk. Your customers will lose confidence in your app, or worse, they may replace it with your competitor's app. To drive engagement and usage, your app must be "reactive"—instantly responding to user interactions in order to maintain UX flow.

In this white paper, we explore the concept of reactive apps and the crucial role of data in the reactive user experience. We'll look at traditional approaches to managing app data flows and solving the challenges presented by the mobile environment. We'll introduce two data models that are designed specifically to support a reactive app architecture. Finally, we'll look at how Realm provides simplified solutions for integrating these data models into any type of modern mobile app.

Modern Apps are Reactive Apps

The term "reactive" is the perfect descriptor for the type of experience that mobile users have come to expect. Reactive applications feel snappy and "alive," always ready to react to user commands, always ready to handle data—regardless of network availability. Such apps provide a consistent, predictable, immediate response. They return results in realtime, providing the user with a sense of instant gratification.

When designing your app, the following principles will help you deliver a reactive user experience:

Immediate UI - take an offline-first approach to ensure that nothing prevents your UI from updating, regardless of network conditions. For example, a messaging app may have an outbox feature that stores messages offline to send later. This will drive continuous use of your app and create the fluid, responsive user experience that will delight your users.

Automatic synchronization across devices - ensure user data is shared immediately, whether it be with your user's own portfolio of devices or with their network. For example, a messaging app may use an indicator feature that informs all users in the chat that someone is typing in realtime. For your user, this feels like their data follows their thoughts and actions, instantly accessible at any time.

Push notifications - utilize messaging services to add value to your app's experience by keeping the user informed of important information. For example, a ride share app may send a notification to alert the user that the car has arrived. Such notifications not only help to keep the user engaged with your business, but they can become an integral part of a great UX.

Apps that are fundamentally designed to be reactive are in a much better position to drive user preference, engagement, and sustained usage. They also enable

sophisticated features, such as multi-user collaboration or data analytics. From the user's perspective, reactive apps are no longer simply best-in-class. They have become the new standard.

The concept of reactivity goes beyond user experience and impacts a wide variety of apps and use cases. Reactive apps also enable fast, data-driven business process across internal systems and device networks. For example, in a multi-app IoT architecture, reactive apps handle data flows in realtime between a network of connected devices, resulting in harmonious system performance. A reactive ticketing app may be constantly updating ticket data in the background, so that the user always sees the latest pricing and availability. In a multitude of scenarios like these, the end user may never perceive such reactivity directly. Yet clearly, both user and business benefit enormously from fast data flows that keep both parties updated and in sync.

Challenges with Developing a Reactive App

Developers face numerous challenges when designing and building a reactive application. Some are specific to the mobile environment, others arise from traditional development approaches that fail to support a truly reactive user experience. Let's take a look at three key hurdles.

Unpredictable Mobile Networks

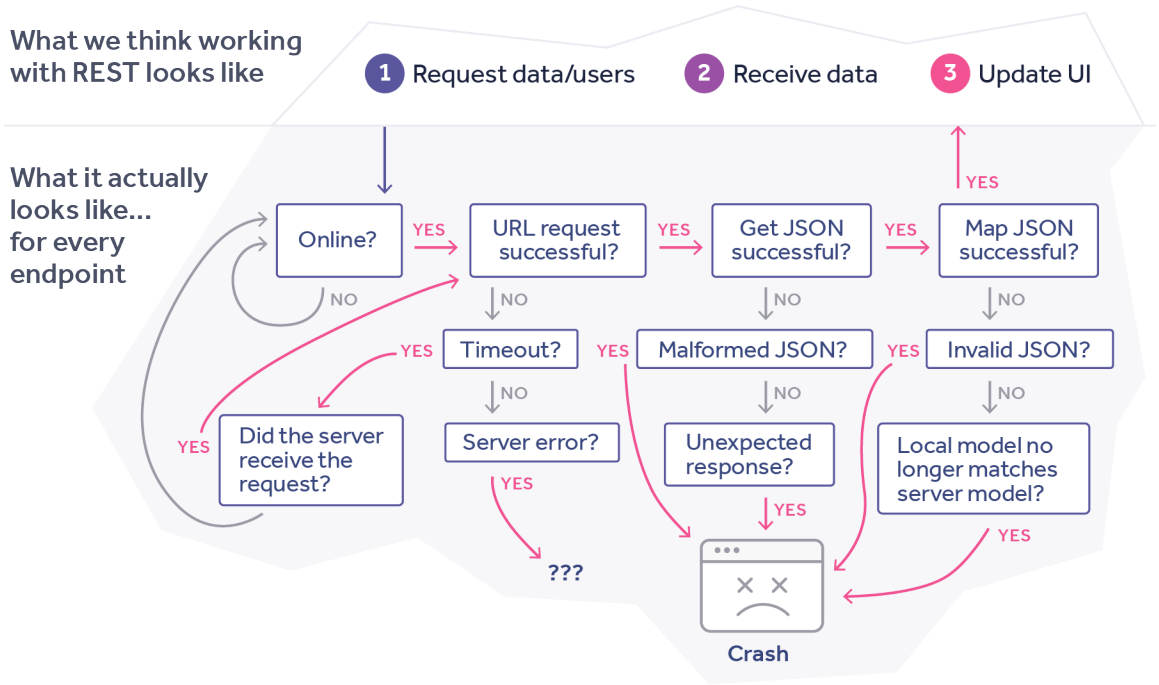
Mobile apps experience a variety of bottlenecks when attempting to transfer data over the network. Latency is inherent in mobile networks, and consistent connectivity is never guaranteed. Even in today's urban environment, networks experience congestion, spotty coverage, and intermittent fluctuations. If a connection drops while transferring data, the user may experience data loss, a broken UI, or long wait times to refresh or update data. To provide a reactive UX, it is essential to consider offline use cases and design apps that enable core functionality to persist, regardless of network conditions.

With the offline-first approach, app developers address the full spectrum of possible network failure scenarios, as well as data conflicts that may arise with offline use. To ensure app resiliency, the traditional approach is to write massive amounts of code to handle such complexity, which results in bloated code that is more challenging to test, debug, and maintain. This increases the time and resources needed to develop the app, and it also distracts the development team from working on features that provide core value to the business.

Cumbersome REST APIs

Along with network issues, REST APIs also impede app responsiveness. Like web developers, mobile app developers favor REST to build APIs that handle data transfer. By nature, APIs follow a request-response-update pattern, which is heavily dependent on the speed and efficiency of the mobile network. The connectivity issues discussed above can delay or disrupt the transfer process, producing a painfully slow user experience. Moreover, intermittent connectivity or transaction conflicts can lead to API call timeouts or data transfer failures that may result in app crashes.

THE REST ICEBERG



For developers, REST APIs in the mobile context are brittle and cumbersome. They require significant development time and effort to ensure resiliency. Developers must write code that handles every aspect of the transfer request, including converting data between multiple formats. More code is needed to transfer data efficiently and prevent excess or duplicate data loads. In addition, when building cross-platform apps, developers must write custom networking code for every supported platform. Finally, REST APIs must be maintained, especially if they are impacted by changes to backend systems. With the API approach, data transfer can easily become a team's biggest focus—and biggest headache.

Complex Legacy System Integration

For apps that share data with backend legacy systems, even more factors can potentially slow response times. The app speed is naturally constrained by the speed of the legacy system itself. In many cases, legacy systems use outdated APIs or protocols which make them difficult to integrate into apps written in modern app languages.

Again, developers must write code that handles all aspects of the transfer, all failure scenarios between systems, and all data conversion. In large organizations, front-end developers often must request and wait for any API changes needed from backend teams, or conversely, respond to app crashes due to unexpected updates to the APIs. App developers will often implement workarounds that compounds the complexity and adds more code to test and maintain—more API calls, more opportunities for failure.

Characteristics of a Reactive Architecture

Behind a reactive user experience is an architecture that is flexible, scalable, and well-equipped to handle change and failure. The principles of reactive programming guide best practices for developing a modern app architecture. Keeping these in mind will help you build effective, consistent reactivity in your app.

There are four key characteristics of a reactive app architecture:

Responsive - your app is able to respond instantly to any user interaction, whether it be inputting data for processing or storage, receiving data from backend systems, or rendering UI components. Response times are fast and predictable, inspiring user confidence in your app.

Resilient - your app responds appropriately to failures stemming from external dependencies, such as network connectivity, and still provides a good user experience. For example, your app enables core offline use cases and transfers data when connectivity is restored.

Scalable - as usage grows, your app architecture is able to scale to keep pace with user demand. Your app and backend server can handle ever-increasing data loads, and still provide the same responsive user experience to your entire user base.

Message Driven - at the heart of your responsive, resilient, and scalable app is asynchronous messaging. Your system uses messaging to push data to your app or trigger server-side events that respond to data changes coming from your app or backend systems.

Choosing the Right Data Solution

At the heart of your reactive architecture is a data model that supports a lightning-fast flow of data between your app and other apps and systems. The right data model will reduce development complexity for your team, and make it easier to overcome the challenges posed by the mobile environment.

Realm takes a fundamentally different approach to data storage and synchronization in order to provide a much faster, richer data experience than standard models. Depending on your app's needs, Realm's data model is expressed in two distinct, but interconnected, data handling solutions.

The first is client-side persistence using the Realm Mobile Database, which can be used as a stand-alone ingredient in your reactive architecture. It will enable many reactive use cases, but it is also impacted by the capabilities of your stack. Your app experience may benefit greatly from simply giving it fast access to data stored locally.

Realm also offers a full data synchronization platform, which provides advanced data handling capabilities, such as syncing data in realtime with backend systems and/or other clients, or running business logic triggered by data events. It includes the Realm Mobile Database as one of its components, but the platform approach will give you the most robust version of Realm's reactive capabilities. Let's take a deeper look at how each solution supports a reactive app experience.

Client-Side Storage

The bulk of mobile application data is traditionally stored on a server or other system that is external to the app itself. The app persists some data locally, however to render the full UI or provide a complete user experience, the app must retrieve data from external sources. For the user, a network-dependent UX can be unpredictably slow, disruptive, and ultimately frustrating. The first step in building a reactive architecture is to implement a data model that provides client-side data storage.

Realm Mobile Database

Realm removes most causes of latency within the traditional database model by embedding a database on device that stores data locally. The Realm Mobile Database is a full-featured database that can be queried, filtered, interconnected, and persisted. It is lightweight and highly performant, capable of handling very large data loads and running queries in fractions of a second. Realm's zero copy policy ensures that no excess or duplicate data is transferred into memory—only the data that your app needs at any one moment. As an alternative to SQLite and Core Data, Realm Mobile Database is much faster than an ORM, and often faster than raw SQLite.

Realm effectively supports your reactive architecture—connect your app's UI to Realm and data changes will appear automatically—and allows you to safely access the same data concurrently from multiple threads, with no crashes. Moreover, Realm's client-side model makes it far easier to serve offline use cases, as most, if not all, of the data that your app needs to function is already on device when connectivity is lost.

Data as Objects

Realm stores data as native objects in a format that is easily consumed by a mobile app using a language binding of the developer's choice. This means that developers do not need to write additional bandwidth-consuming code to serialize or deserialize data, or deal with complex object relational mapping. Realm objects are always "live," which means that they always have the latest data. When data changes, notifications enable your app to instantly react and update the UI accordingly.

With the traditional model, the database administrator sets the data model in the back-end database. With Realm's simple object-oriented structure, the developer sets the data model in code, which effectively means that your database is your data model. In addition, Realm is cross-platform, so the same data layer can be shared across all your apps running on any major mobile platform.

Simplified Development

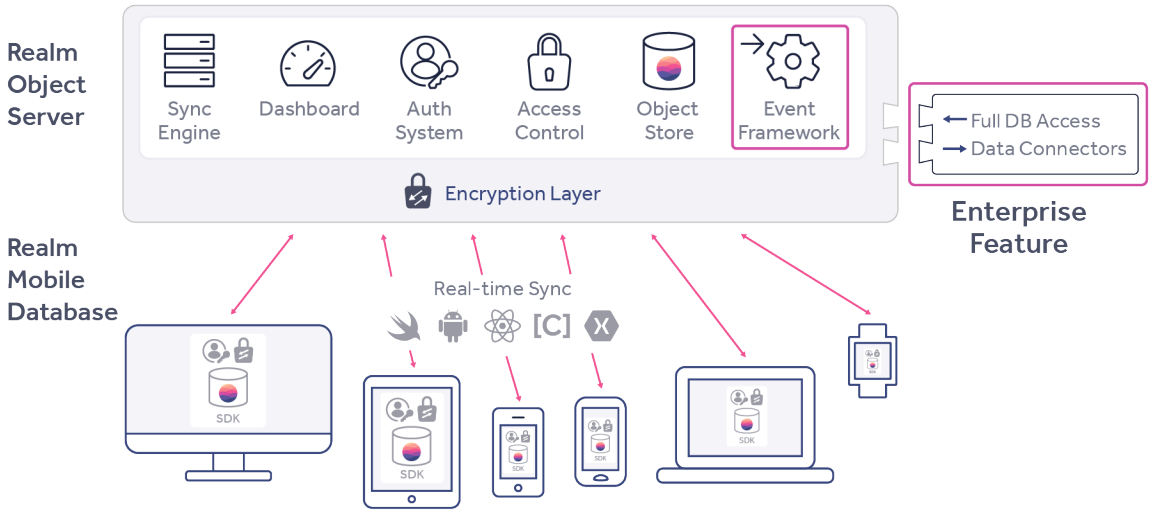
To date, there have been more than 1 billion over installs of Realm Mobile Database worldwide, and this base is growing rapidly. Realm is simple to implement and developers can get started in minutes. Your app needs no APIs to transfer data and no excess code to translate data or ensure resiliency—less code to write means less code to test, debug and maintain. Realm shortens your development cycles and frees your team to focus on building features that add core value to your users and your business. In addition, the Realm Mobile Database is available for major mobile languages, such as Swift and Objective-C (iOS), Java (Android), C# (Xamarin, .NET), and Javascript (React Native and Node.JS). See our current language support for more information.

Data Synchronization Platform

Data is traditionally retrieved, updated, and synchronized using a call-response protocol, such as a REST API. This approach is cumbersome for developers, requiring significant code to address the complexities of resiliency, API calls, data conversion, legacy system integration, and more. In addition, REST is not designed for pushing data and developers must create workarounds to enable some of a reactive app's more sophisticated features. The cornerstone of your complex, reactive architecture is a full-featured platform that manages all data flows and synchronization across your entire client install base and relevant backend business systems.

Realm Mobile Platform

Realm simplifies application architecture and greatly reduces complexity for developers by removing the dependency on APIs to handle data flows. The Realm Mobile Platform includes two main components: the Realm Mobile Database (as discussed above) and the Realm Object Server. The server functions as a middleware component in the mobile app architecture and manages data synchronization, event handling, and integration with legacy systems. Realm's approach ensures cross-platform compatibility across your entire app install base. This unified model means that your development team designs and builds your app's data model only once, and then simply rolls it out across your app variants.



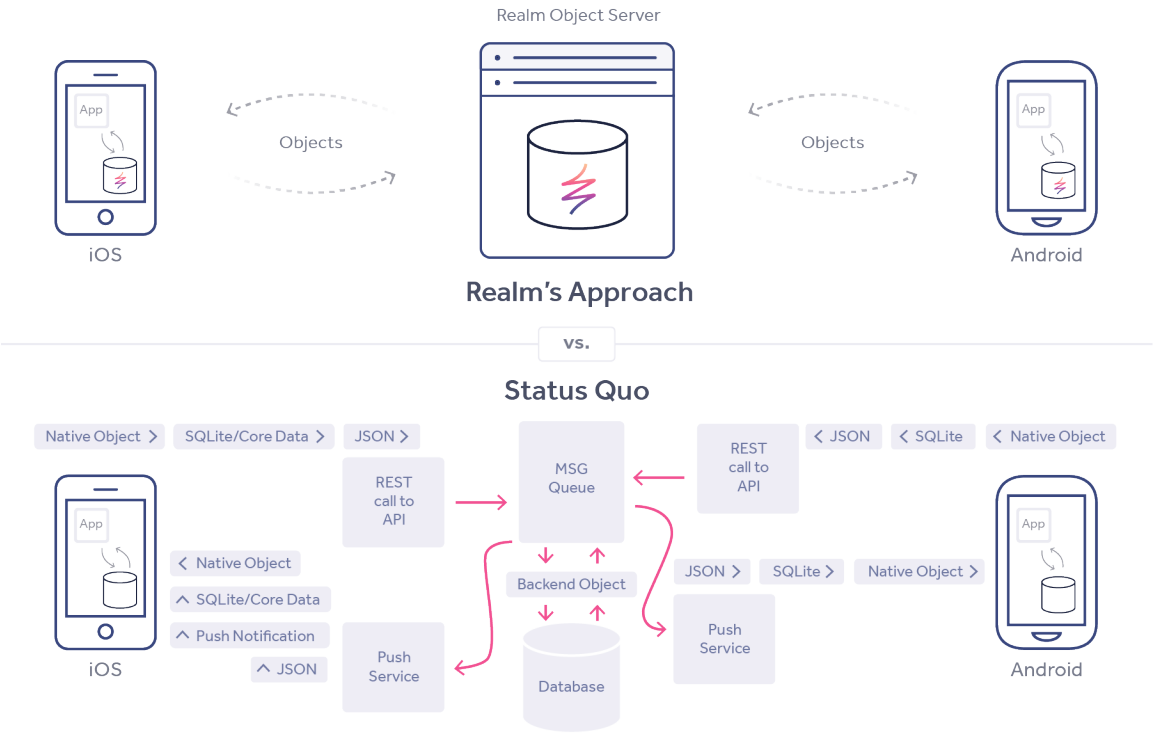
Realtime Data Synchronization

Based on the concept of persistent shared state, data is stored as objects in the app's native language. These objects are mirrored between the Realm Mobile Database on the client and the Realm Object Server, and the two communicate seamlessly using a realtime data sync channel. In this model, no data translation code is needed. Data changes coming from the client are immediately synced with the same data objects on

the server, as well as with all other clients that share those objects. Conversely, data changes on the server can be pushed to all relevant clients. The platform handles any complexities, such as data conflict resolution or network connectivity issues, freeing the developer from these time-consuming tasks. The Realm Object Server can efficiently and simultaneously sync data across 1m+ devices and automatically resolve conflicts—all in realtime.

Live Objects as the new APIs

In Realm's object-oriented approach, every object is "live," meaning that the variable is always the correct representation of the underlying data. Realm attaches "listeners" to data that react (or file callback methods) when that data changes. Realm automatically initiates data synchronization and updates the object in realtime. Data sync is fast and efficient—only incremental or additive changes are transmitted, not entire objects. The entire data flow is handled seamlessly by Realm, with no additional networking code needed. With live objects, your users always have the latest data at hand and will appreciate a smooth, highly performant app experience.

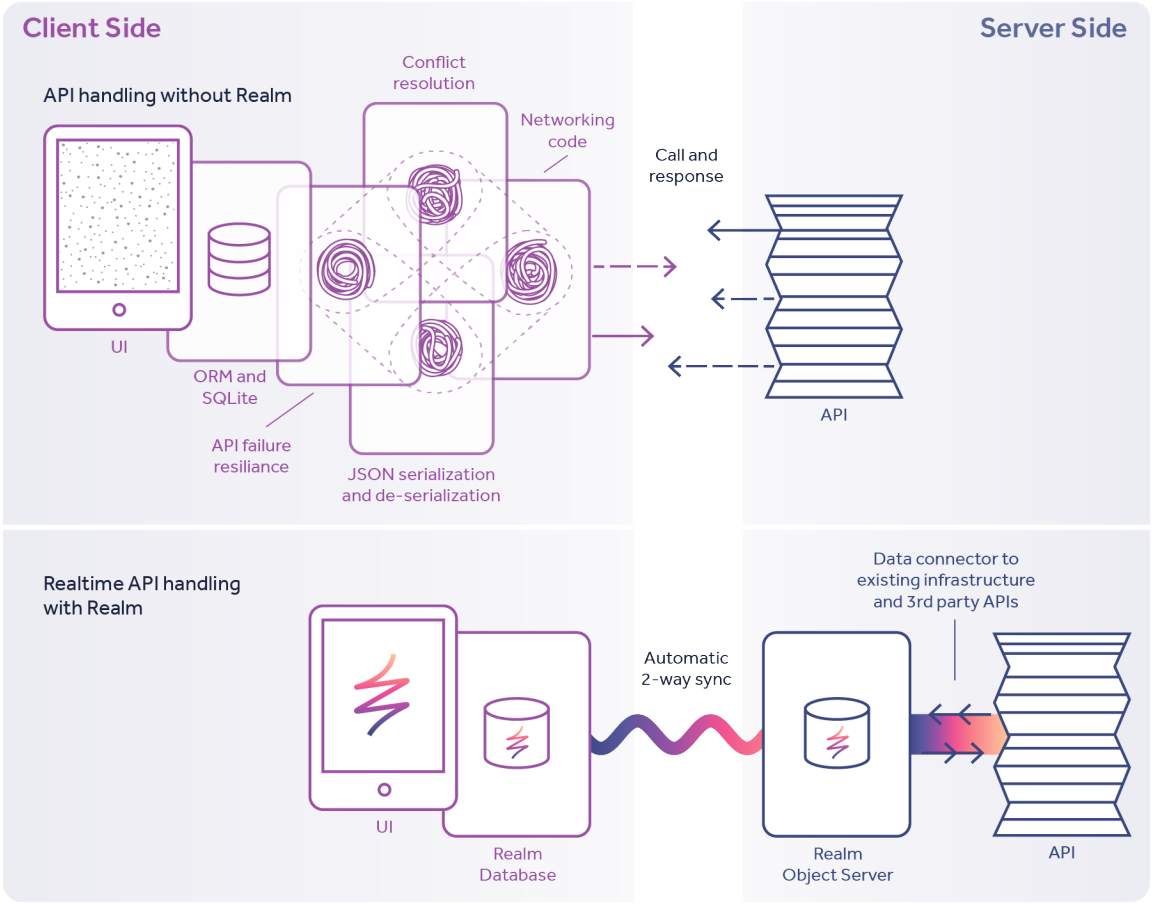


Event Handling Framework and Serverless Logic

Realm provides an event handling framework that can trigger server-side logic whenever an object changes. Using the Realm Node.js SDK, developers write business logic that governs how data flows between apps and and the Realm Object Server. When data changes on the client side, it gets synced to the server, which then executes code to handle the change. Event handling also enables the Realm Object Server to trigger data flows between your app and legacy APIs and systems. Using Realm Functions, you can integrate "serverless" computing principles into your app architecture and write discrete functions for each specific trigger or other server-side features. Realm removes the need to add another endpoint to a server and then write serialization and networking code to connect to it.

Mobilizing Legacy APIs

Many apps need to surface core business data within the app and funnel user-generated data back into legacy systems of record. The Realm Object Server provides a single place to efficiently manage all communications, including legacy API transactions, without risk of disruption from network issues. Developers no longer have to wrestle with the complexity of supporting multiple legacy APIs across multiple mobile platforms and an unstable network. Using the Realm Node.js SDK, developers utilize various API connectors to establish connections between the Realm Object Server and legacy systems. No extra code is needed on the client side. Data flows from the legacy system into objects on the server, which then syncs that data with the same objects on the client.



Easy to Implement

Currently, over 100k+ mobile developers actively use Realm to build apps. One of the reasons why Realm is so popular with developers is that it is exceptionally flexible and easy to use. The Realm Mobile Platform can easily fit into your existing system architecture as a lightweight, modular component, and it is versatile enough to integrate under multiple architectural patterns. Based on your business needs, you can choose to deploy Realm Object Server on existing on-premise servers behind your firewall, in your private cloud, or on your AWS or Azure instances in the public cloud.

Learn more about the Realm Data Model and its features

Reactive App Use Cases

With a reactive architecture at its core, your application is well-positioned to deliver a wide range of sophisticated features and functionality. The following are four common use cases that are only viable in a reactive app.

Realtime Collaboration

Many of today's most successful mobile apps allow users to share and interact with data in realtime. For example, messaging apps like WhatsApp depend on continuous conversation between users. Uber visualizes the position and movement of their available fleet. Instagram enables instant photo sharing and engagement. For these apps, immediacy is fundamental to the app experience itself, as well as to the mobile business.

Designing a reactive mobile collaboration experience requires a data model that enables realtime synchronization. The Realm Mobile Platform uses "live objects" that automatically sync data updates across devices and a central server. This means that if a user inputs or changes data in an iPhone app, that data is synced in realtime with any other app clients that depend on, or have access to, that data, whether they be iOS or Android. Realm's automatic conflict resolution system ensures predictable merging of record data, even with multiple writers. The Realm Mobile Platform simplifies data synchronization and data handling, which frees your developers to focus their expertise on building an exceptional collaborative UX.

Learn more about using Realm to build realtime collaboration into your apps

Realtime Feedback

Apps that provide useful feedback to mobile users in realtime are powerful tools for engagement. Such apps capture user data, process or analyze it, and return insights back to the user. For example, Realm customer Arccos captures data through motion sensors embedded in a golf club to analyze a player's strengths, weaknesses, and trends, delivering feedback through the mobile app in realtime. Mapping apps use GPS and traffic data to track a user's location, recommend the fastest route, and estimate the arrival time. Realtime feedback is integral to a user's flow and enhances the user's experience in that moment.

To support the feedback process, apps rely on the client-side storage capabilities of Realm Mobile Database. By accessing data locally, your app greatly reduces the opportunities for latency and failure that would otherwise happen using the call-response-update API protocol over the mobile network. Your developers can focus on building less API-related code, such as those that transfer data between your client and your analytics engine. Or, they can bypass the need for APIs altogether by using the Realm Mobile Platform, which would handle all data transfer, as well as legacy system integration. Realm also supports a good offline experience by allowing the app to continue to function offline, and automatically transferring changes once it's connectivity is restored.

Learn more about how Arccos uses Realm to power their connected golf experience

Realtime Business Processes

Customers are not the only stakeholders who expect reactivity in apps. Internal-facing apps depend on a reactive architecture to enhance employee productivity or enable timely business processes. For example, a field worker management app needs to track the task status and geolocation of its workforce, and match available workers with a new tasks as they arise. A water management app relies on data coming from sensors that measure water flow and pressure. Realtime data allows engineers to monitor system performance and immediately alert maintenance crews of any potential problems. In such cases, the health of the business is dependent on instant access to data needed to trigger appropriate business processes.

Apps with complex moving parts need a streamlined solution for handling data from numerous sources in realtime. The Realm Mobile Platform features an event handling framework that allows developers to easily write business logic that is triggered by data changes and executes specific processes. The platform APIs also provide easy connectors from the server to backend databases, giving developers a single place to manage legacy system communications. All data is stored on the client and server using the same object model, which means that no data translation code is needed. Moving much of the app's complexity to the platform will simplify your app architecture and make it easier build in reactivity from the ground up.

Learn more about Realm's event handling capabilities and platform APIs

Realtime Notifications

More and more apps are using notifications to inform the user of an important event or action needed. For some apps, this is a core aspect of the UX. For example, a meeting scheduler will notify users that their upcoming meeting has been cancelled. An alarm system app will alert the user if a door or window has been left open. Or an airline app will keep users updated on the estimated departure time for their delayed flight. In these cases, the app is reacting to data changes happening on the backend, triggering the business to pro-actively reach out to its users to keep them informed and engaged.

Developers who turn to REST APIs to enable push notifications in their app will be forced to create a workaround. REST is not designed for pushing data, and there's no ability to push a REST call out. A much simpler approach is to manage notifications through the Realm Mobile Platform. The platform's event handling framework allows developers to create logic that pushes out notifications to clients as needed. There is no need to force the app to pull messages or work with external mechanisms, such as Apple's push system. Your notification processes are centralized on a single server, giving you one place to build and manage them across all mobile platforms.

Learn more about how Realm handles notifications

Learn More About Realm

Building a reactive application does not have to be a daunting task. You can ease the burden on your development team by integrating Realm's data model into your app architecture. Realm takes care of all the heavy lifting around data flows, so your team can keep code lean, performant, and easier to maintain. With less code to manage, your team is free to do what they do best: build a great user experience.

To learn more, we encourage you and your team to try both our database and platform:

Realm Mobile Database is free, open source, and available in a range of languages.

Realm Mobile Platform offers a range of options. Download the free Developer Edition, the 60-day trial of the Professional Edition, or contact us for a free demo of the Enterprise Edition.

www.realm.io

Realm HQ
148 Townsend Street
San Francisco, CA 94107 USA

Founders House
Njalsgade 21G
2300 Copenhagen S Denmark

Build better apps faster.