**International Journal of Management, IT & Engineering**
Vol. 13 Issue 12, December 2023
ISSN: 2249-0558 Impact Factor: 7.119
Journal Homepage: http://www.ijmra.us, Email: editorijmie@gmail.com
Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gate as well as in Cabell's Directories of Publishing Opportunities, U.S.A

# Offline First Architecture in Real-Time Applications

**Prateek Sharma**

## Abstract

This article provides an in-depth exploration of offline-first architecture, an important architecture for enhancing user experience in various domains such as warehouse management, healthcare, and retail. It highlights the need for this architecture in scenarios where network reliability is inconsistent and covers essential aspects of offline storage, including data types stored, synchronization methods (Manual, Push-based, or Pull-based), and the importance of security for locally stored data. It addresses the challenges and solutions in syncing local data with server data, ensuring a seamless balance between offline and online operations. The article also discusses various conflict resolution strategies in offline-first architecture, such as Last Write Wins, Manual Conflict Resolution, Object Versioning, and CRDTs. Each method is examined for its effectiveness in different scenarios, underscoring the importance of choosing the right strategy based on specific use cases and product requirements.

***Keywords:***

Offline First Architecture,
Data Synchronization,
Conflict Resolution,
Offline Storage,
Network Reliability.

***Author correspondence:***

Prateek Sharma,
Masters in Computer Engineering
Email: sharmaprateek10@gmail.com

## 1. Introduction

In the current world, offline-first architecture is increasingly important across various domains, such as warehouse management systems, healthcare applications, retail inventory tracking, etc. Despite widespread internet access, network reliability still varies greatly from one location to another. Many real-time applications depend heavily on network connectivity and can struggle with responsiveness during network disruptions. Therefore, adopting offline-first architecture is crucial for these applications to ensure a better user experience and maintain responsiveness, even when network issues arise.Additionally, offlinefirst architecture offers additional benefits, including optimization of battery and data usage, as well as enhancing the overall robustness of applications.

Offlinefirst architecture needs thoughtful planning for both the client and server sides to handle network disruptions. On the client side, this means creating local storage for saving user activities and app assets when there's no network. It also includes managing how data synchronizes with the server once the network is back. On the server side, the focus is on resolving conflicts that happen when multiple users have edited the same item while offline. There are several approaches server can implementto resolve conflicts like last write wins, versioning of entities, human intervention etc.

## 2. Offline First Architecture

In offline first architecture, system designensures that application is unaffected by the network interruptions and user can effectively use the application with or without network. This is achieved by having a local offline storage which can act as a proxy if the client application does not have network connectivity. This offline storage can hold different types of data:

1. **Application Assets:**Essential elements like language translations, images, and other assets that don't change often are stored locally. This means the app doesn't need to keep asking the server for these resources.
2. **CachedServer Responses:**The offline storage can save server responses. This cache helps the application function offline and updates when there's network access again.
3. **Request Queue:**The application keeps a list of user requests or API calls (like write operations) in this offline storage. These requests are sent to the server asynchronously once the network is back.

Depending on the requirements of an application and its business context, various databases can be utilized for offline storage, including options like SQLite, CouchDB, and more. The selection of a database often hinges on the specific functionalities needed. For example. SQLite is known for its lightweight and reliable structure, making it ideal for mobile and embedded applications. CouchDB, on the other hand, lets data flow seamlessly between server clusters to mobile phones and web browsers, enabling a compelling offline-first user experience.

One of the most important aspects of offline storage is how data is synced with the server. There are a few data synchronization approaches, and they are discussed in detail in the next section. Additionally, another important aspect of offline storage is security – since user and application data is stored locally, the application needs to ensure that it is encrypted and signed so it can't be tampered with. Additionally, when the app sends data from offline storage to the server, it should verify that the authentication tokens for these requests are valid.

In offline applications, conflicts can arise when multiple users edit the same data while offline. For instance, consider a social media app where a user replies to a comment on their post without a network connection. If another user has already updated or deleted that comment, this leads to a conflict because the data has changed. The server handles these conflicts. It first checks for any discrepancies in the data received from clients. If conflicts are found, the server applies a conflict resolution strategy (which is discussed in later sections) to ensure data consistency. After resolving these conflicts, the server updates all clients with the latest, unified data view, allowing them to refresh their local storage.

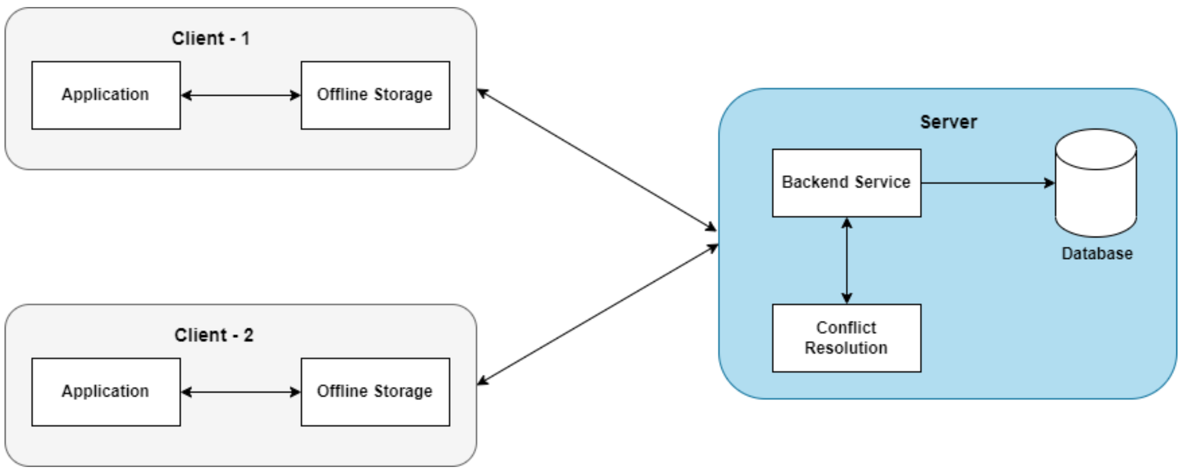

Figure 1. *Offline First Reference Architecture*

Figure 1 shows the reference architecture for an offline first application, illustrating how it incorporates offline storage and conflict resolution components.

## 3. Data Synchronization

Data synchronization, the task of aligning local offline storage with server data, is crucial in offline-first applications. It manages how and when local data is updated or sent to the server as soon as the app connects to the internet. This process ensures a smooth balance between offline and online data. Typically, there are three primary methods for an application to sync data with the server: Manual, Push-based, or Pull-based.

### 3.1 Manual Synchronization

With manual synchronization, the user starts the process of updating data with the server. This allows users to decide when to refresh the contents of offline storage, like app assets or cached resources. This mode is especially helpful for devices that are often disconnected from the network. Users can choose to synchronize data whenever they have a stable network connection.

### 3.2 Pull-based Synchronization

In pull-based synchronization, the applicationdetermines how often local data is synchronized with the server. Applications can use a pre-determined time-based schedule (for example, every hour) to refresh the local data.This method is generally simpler to set up and maintain. However, it's important to consider certain aspects. For example, the application might repeatedly try to update data that hasn't changed, or it could try to fetch data when there's no internet connection. Also, this approach can lead to additional API calls to the server, which need to be managed effectively.

### 3.3 Push-based Synchronization

In the push-based synchronization model, the server takes the lead in sending updates to the application, instead of the application constantly checking for them. Initially, the application fetches the needed data at startup. After that, the server notifies the app about any new updates. This means that as soon as the app is online, it automatically receives updates from the server, keeping its local offline storage up to date. This method enables the app to function in offline mode for extended periods while conserving resources. However, for this model to work, the server must be equipped to handle a publish-and-subscribe system.

A real-time application can incorporate all three synchronization models, tailoring them to fit specific features. Depending on the product's needs, the application can choose the most suitable mode for each use case. For instance, it might use pull-based synchronization for static image assets, and push-based synchronization for updates to server data stored in its cache.

Additionally, there are databases designed to facilitate smooth data replication between the application and the server. For instance, in a system where PouchDB is used as the client-side database for offline storage and CouchDB as the server-side database, PouchDB can automatically sync any local changes with the CouchDB server. Similarly, PouchDB can effortlessly retrieve updates from the server, keeping its offline storage current without requiring the application to manage synchronization processes.

All the synchronization models discussed here have their own advantages and disadvantages, and selecting the appropriate one depends on the specific product requirements and the existing client/server infrastructure.

## 4. Conflict Resolution

A conflict arises when the local data on an application doesn't match what's on the server. Take this example: two users, U1 and U2, are working on the same task. U1 is online and manages to communicate with the server to accept the task. Meanwhile, U2 is offline due to a network issue but also accepts the task, which is temporarily stored in their offline storage queue. Once U2's application reconnects to the network, it attempts to accept the task, only to find that it has already been accepted by U1. This situation leads to a conflict because both users were trying to complete the same task. The resolution of such conflicts depends on how the application and server are set up to handle them.

In an offlinefirst architecture, clients will occasionally be out of sync with the server, leading to conflicts. It's the server's responsibility to implement a conflict resolution strategy and act as the ultimate source of truth. Below, we discuss various strategies for resolving these conflicts.

### 4.1 Last Write Wins

In this method, each time an application sends data to the server via an API call, it includes a timestamp indicating when the action occurred. The server then uses this timestamp to manage conflicts. If a conflict arises, the server favors the change with the most recent timestamp, discarding the older one. For example, imagine two users accept a task at times T1 and T2, respectively, with T2 being later than T1. In an offline mode, when both users eventually connect to the network and their requests reach the server, the task will be assigned to the user associated with the T2 timestamp, as their action was the latest.

Though this approach is straightforward for managing conflicts and is widely used in many applications, it does have a drawback of data loss. Therefore, it's crucial to carefully evaluate the product's requirements to determine if this method is suitable for the specific use case.

### 4.2Manual Conflict Resolution

In this method, conflicts are resolved by users according to specific business rules. It's typically applied when conflicts are infrequent and need explicit manual review. For instance, in inventory management system, if two users update the audit status of a product differently, an administrator would be notified to manually resolve the discrepancy.

### 4.3Object Versioning

In this version-based approach, data on the server is assigned a new version each time it's successfully updated, while also retaining its previous version as a 'parent'. The full version history is maintained on the server. A conflict is identified when two updates share the same parent version. The server then takes charge of resolving these conflicts, possibly using established strategies like 'last write wins' or manual resolution. For example, in a warehouse management system, if two users accept the same task while offline, both attempting to update from version N to N+1, the server recognizes the conflict because they share parent version N. The server might then direct the conflict to be resolved manually by a user. PouchDB and CouchDB databases utilize a version-based approach for conflict resolution, maintaining versions on both the application and server sides. In case of conflicts, CouchDB typically selects a winner through a predefined algorithm. However, manual resolution of these conflicts is also an supported.

### 4.4CRDT (Conflict free replicated data types)

With a Conflict-Free Replicated Data Type (CRDT), concurrent updates can always be merged or resolved without conflicts, eliminating the need for a central decision-maker. The main strategy is to transform all edit operations into commutative ones, meaning their sequence doesn't affect the outcome. Practically, this approach resolves issues with changes that arrive 'out of order'. This is because there's either no set order for changes made on different copies of the data, or, even if there is a correct order, it's not always possible to ensure every change reaches all copies in that exact sequence.

## 5. Conclusion

This article not only sheds light on the technicalities of offline-first architecture but also presents a clear roadmap for implementing effective synchronization and conflict resolution strategies. As digital applications continue to evolve, understanding and leveraging offline-first architecture will be key to delivering seamless and resilient user experiences, particularly in environments where network connectivity is a challenge.

## References

[1] Mohammadreza Sharbaf, Bahman Zamani, and Gerson Sunyé, "Conflict Management Techniques for Model Merging: A Systematic Mapping Review", Software and Systems Modeling, inPress,nPress. hal-03787436,Sep 2022

[2] Mehdi Ahmed-Nacer, Claudia-Lavinia Ignat, Gérald Oster, Hyun-Gul Roh, and Pascal Urso, "Evaluating CRDTs for Real-time Document Editing",11th ACM Symposium on Document Engineering, pp.103–112, Sep 2011

[3] Hasura. (n.d.). Design guide to offline-first apps. Retrieved from https://hasura.io/blog/design-guide-to-offline-first-apps/

[4] Ably Realtime. (n.d.). CRDTs & the Quest for Distributed Data Consistency. Retrieved from https://ably.com/blog/crdts-distributed-data-consistency-challenges#what-are-crdts