Offline-First Strategy

Offline-First Strategy

> **Project:** {{PROJECT_NAME}} **Version:** {{VERSION}} **Date:** {{DATE}}
> **Author:** {{AUTHOR}} **Status:** Draft | In Review | Approved **Reviewers:** {{REVIEWERS}}

Document History

Version	Date	Author	Changes
0.1	{{DATE}}	{{AUTHOR}}	Initial draft

---

1. Offline Capability Requirements

Feature	Offline Support	Priority	Notes
View cached content feed	Required	P1	Last 50 items
Create draft (saved locally)	Required	P1	Sync when online
Search (local cache only)	Partial	P2	Degraded — no remote results
User authentication	Not required	—	Login requires network
Push notifications	N/A	—	Requires network by nature
File uploads	Queue	P2	Upload when network returns
`{{FEATURE}}`	`{{Required/Partial/Not required}}`	`{{P1-P3}}`	`{{Notes}}`

Offline minimum viable experience:

>

> TODO: Define what the user should see/do when completely offline — blank screen, cached data, or read-only mode.

---

2. Local Storage Architecture

2.1 Database (Structured Data)

Selected database: `{{WatermelonDB | SQLite (expo-sqlite) | Realm | TinyBase}}`

Rationale:

> TODO: Explain why this DB was chosen (query capability, sync support, performance, bundle size).

Schema overview:

```
-- Example schema — expand per domain

CREATE TABLE users (
  id TEXT PRIMARY KEY,
  name TEXT NOT NULL,
  email TEXT UNIQUE NOT NULL,
  avatar_url TEXT,
  synced_at INTEGER,
  updated_at INTEGER NOT NULL
);

CREATE TABLE posts (
  id TEXT PRIMARY KEY,
  user_id TEXT REFERENCES users(id),
  title TEXT NOT NULL,
  body TEXT,
  status TEXT DEFAULT 'published',
  is_local_draft INTEGER DEFAULT 0,
  synced_at INTEGER,
  created_at INTEGER NOT NULL,
```

```
  updated_at INTEGER NOT NULL
);

CREATE TABLE sync_queue (
  id TEXT PRIMARY KEY,
  entity_type TEXT NOT NULL,
  entity_id TEXT NOT NULL,
  operation TEXT NOT NULL, -- 'create' | 'update' | 'delete'
  payload TEXT NOT NULL,   -- JSON
  retry_count INTEGER DEFAULT 0,
  created_at INTEGER NOT NULL
);
```

TODO: Define full schema for all entities.

2.2 File Storage

Type	Location	Max Size	Eviction
Downloaded images	`{{FileSystem.cacheDirectory}}/images/`	200 MB	LRU on cache full
Downloaded documents	`{{FileSystem.documentDirectory}}/docs/`	500 MB	Manual user delete
Queued upload files	`{{FileSystem.documentDirectory}}/uploads/`	1 GB	On successful upload
Temporary files	`{{FileSystem.cacheDirectory}}/tmp/`	50 MB	On app start

Library: `{{expo-file-system | react-native-fs}}`

2.3 Secure Storage

Data	Storage	Reason
Auth token	`{{expo-secure-store}}`	Encrypted, Keychain/Keystore
Refresh token	`{{expo-secure-store}}`	Encrypted
Encryption key (for local DB)	`{{expo-secure-store}}`	Never in plain storage
User preferences	`AsyncStorage`	Non-sensitive

Rule: Anything accessed without a password must NOT be in secure storage (breaks biometric auth flow).

3. Sync Protocol Design

```
sequenceDiagram
    participant App
    participant LocalDB
    participant SyncQueue
    participant API

    Note over App,API: Online sync cycle

    App->>LocalDB: Read local data (immediate)
    App->>SyncQueue: Queue local changes
    App->>API: Push: POST /sync/push {changes}
    API-->>App: Server-applied changes + conflicts
    App->>LocalDB: Apply server changes

    App->>API: Pull: GET /sync/pull?since={timestamp}
    API-->>App: Remote changes since last pull
    App->>LocalDB: Merge remote changes

    Note over App,API: Offline scenario

    App->>LocalDB: Read cached data
    App->>SyncQueue: Queue changes (persisted)
    Note over SyncQueue: Waits for connectivity

    Note over App,API: Reconnect

    SyncQueue->>API: Drain queue — push all pending
    API-->>App: Conflict resolution
    App->>LocalDB: Merge resolved state
```

3.1 Sync Strategy

Approach: `{{Bidirectional delta sync}}`

Property	Value

Protocol	REST + `{{GraphQL subscriptions / WebSocket for live}}`
Push endpoint	`POST /sync/push`
Pull endpoint	`GET /sync/pull?since={unix_ms}&entities={list}`
Sync identifier	Per-entity `updated_at` timestamp (server clock)
Pull delta	Only records changed since last sync cursor
Batch size	Max 100 records per push, 200 per pull

3.2 Conflict Resolution

Entity	Strategy	Rationale
User profile	Last Write Wins (server wins)	Single-user edit
Post drafts	Last Write Wins (client wins on local draft)	User owns draft
Settings	Merge (union)	Non-conflicting fields
Counters (likes, views)	Server-side CRDT	Concurrent increments
`{{Entity}}`	`{{LWW / CRDT / Manual / Server wins}}`	`{{Reason}}`

Conflict detection: Compare `updated_at` + server-assigned `version` counter.

Manual conflict flow (when required):

1. Server returns `409 Conflict` with both versions
2. App stores both versions in local DB
3. User presented with diff UI to choose version
4. Resolved version pushed back to server

3.3 Sync Frequency & Triggers

Trigger	Action	Conditions
App foreground	Pull sync	Network available
Mutation (create/update/delete)	Immediate push	Network available; else queue
`AppState` change: background → foreground	Full sync	> 5 min since last sync
Network restored	Drain sync queue	Any queued changes
Timer (background fetch)	Pull sync	`{{Every 15 min}}`

Trigger	Action	Conditions
Push notification received	Pull sync for affected entity	Notification type = 'data_update'

3.4 Partial Sync / Delta Sync

- Client stores `last_sync_cursor` per entity type (epoch milliseconds)
- Pull requests include `since` cursor — server returns only changed records
- Deleted records: server maintains soft-delete with `deleted_at` for 30 days
- Client applies deletions, then removes soft-deleted records from local DB

---

4. Sync Queue Management

Queue storage: SQLite `sync_queue` table (survives app restart)

Queue item schema:

```
interface SyncQueueItem {
  id: string;             // UUID
  entityType: string;     // 'post' | 'user' | etc.
  entityId: string;
  operation: 'create' | 'update' | 'delete';
  payload: object;        // Full entity data
  retryCount: number;
  maxRetries: number;     // 5
  createdAt: number;      // Unix ms
}
```

Drain strategy:

1. On network restore: drain queue in FIFO order
2. Batch up to 50 items per push request
3. On error: retry with exponential backoff (1s, 2s, 4s, 8s, 16s)
4. After `maxRetries`: move to dead letter queue, notify user

TODO: Define user notification UX for sync failures.

---

5. Network State Detection & Handling

Library: `{{@react-native-community/netinfo}}`

```
// Network state hook
export function useNetworkState() {
  const [isOnline, setIsOnline] = useState(true);
  const [connectionType, setConnectionType] = useState<string>('unknown');

  useEffect(() => {
    return NetInfo.addEventListener((state) => {
      setIsOnline(state.isConnected && state.isInternetReachable);
      setConnectionType(state.type);
    });
  }, []);

  return { isOnline, connectionType };
}
```

UI behavior per state:

State	UI Response
Offline	Banner: "You're offline — showing cached data"
Reconnected	Banner: "Back online — syncing..." (auto-dismiss 3s)
Slow connection	No extra UI (handle transparently)
Sync in progress	Subtle indicator (not blocking)

---

6. Data Flow: Online vs Offline

```
flowchart TD
    UserAction["User Action"] --> CheckNetwork{Network\nAvailable?}

    CheckNetwork -->|Yes| DirectAPI["Send to API\ndirectly"]
    DirectAPI -->|Success| UpdateLocal["Update local DB"]
    DirectAPI -->|Error| QueueAction["Queue action\n+ optimistic update"]

    CheckNetwork -->|No| QueueAction
    QueueAction --> UpdateLocal
    UpdateLocal --> UpdateUI["Update UI\n(optimistic)"]
```

```
    style UpdateUI fill:#d4edda
    style QueueAction fill:#fff3cd
```

---

7. Testing Strategy for Offline Scenarios

Test Type	Scope	Tool
Unit	Sync queue operations	Jest
Unit	Conflict resolution logic	Jest
Integration	DB read/write with mock network	Jest + in-memory DB
E2E	Full offline → reconnect flow	Detox / Maestro
Manual	Network conditions simulator	`Network Link Conditioner` (iOS), `tc` (Android emulator)

E2E offline test scenario:

1. Open app online — verify data loads
2. Enable airplane mode
3. Perform create/update/delete actions
4. Verify optimistic UI updates
5. Verify actions queued (inspect DB)
6. Disable airplane mode
7. Verify sync queue drains
8. Verify server data matches local state

---

8. Storage Limits & Data Eviction Policy

Storage Type	Soft Limit	Hard Limit	Eviction Strategy
SQLite DB	50 MB	200 MB	Evict records older than 30 days
Image cache	150 MB	300 MB	LRU eviction
Document cache	200 MB	500 MB	User prompt to clear
Total app storage	500 MB	1 GB	Warn user, offer cleanup

Low storage alert: When device storage < 500 MB free, reduce cache limits by 50%.

User-initiated cleanup: Settings → Storage → Clear Cache option.

9. Error Handling & User Feedback

Error	User Feedback	Recovery Action
Sync push failed	Toast: "Couldn't sync — will retry"	Auto-retry with backoff
Conflict detected	Modal: "Update conflict — please resolve"	Manual resolution flow
Queue overflow (>500 items)	Warning banner	Partial push, user notified
Local DB corruption	Alert: "Storage error — please reinstall"	Offer fresh install
Storage limit reached	Alert with cleanup CTA	User clears cache

Approval

Role	Name	Date	Signature
Author			
Mobile Lead			
Backend Lead			
Product Owner			

Revision #7
Created 2026-02-23 12:05:19 UTC by John
Updated 2026-05-25 07:33:08 UTC by John