

Versioning Strategy

Emmanuel BRUNO
Université de Toulon, LIS UMR CNRS 7020
emmanuel.bruno@univ-tln.fr

2026-10-14

Table des matières

1	Versioning Strategy	1
1.1	Semantic Versioning	1
1.2	Version Components	1
1.2.a	Mobile App Version	1
1.2.b	Build Number	1
1.3	Version Lifecycle	2
1.3.a	Pre-Release Versions	2
1.3.b	Development Builds	2
1.4	Version Storage and Access	2
1.5	Versioning of API and Components ..	2
1.5.a	Backend API	2
1.5.b	Database Schema	2
1.6	Version Compatibility	2
1.6.a	Minimum OS Version	2
1.6.b	API Compatibility	2
1.7	Version Bumping Procedures	3
1.7.a	When to Bump Versions	3
1.7.b	Version Bump Process	3
1.8	Release Branches	3
1.9	Version in Documentation	3
1.10	Archiving and Maintenance	3
1.11	Examples	3
1.11.a	Standard Release Pattern Example	3
1.11.b	Hotfix Example	3

1 Versioning Strategy

This document outlines the versioning strategy for the Fitness Tracker application, ensu-

ring clear communication about changes and compatibility across releases.

1.1 Semantic Versioning

We follow Semantic Versioning 2.0.0 (SemVer) for the Fitness Tracker application. Each version number consists of three components:

```
MAJOR.MINOR.PATCH
```

Where:

- **MAJOR**: Incremented for incompatible API changes or significant UI/UX overhauls
- **MINOR**: Incremented when adding functionality in a backward-compatible manner
- **BUILD**: Incremented for backward-compatible bug fixes and minor enhancements

1.2 Version Components

1.2.a Mobile App Version

The mobile app version is displayed to users and consists of the semantic version (e.g., 1.2.3).

1.2.b Build Number

Each build has an incremental build number that increases monotonically regardless of the semantic version:

- Android: `versionCode` (integer)
- iOS: `CFBundleVersion` (integer)

Example: Version 1.2.3 might have build number 45.

1.3 Version Lifecycle

1.3.a Pre-Release Versions

Pre-release versions follow this pattern:

```
MAJOR.MINOR.PATCH-[alpha|beta|rc].N
```

Where: - **alpha**: Early development versions for internal testing only - **beta**: Feature-complete versions ready for user testing - **rc**: Release candidates for final validation - **N**: Sequential number starting at 1 (e.g., 1.2.0-beta.2)

1.3.b Development Builds

Daily development builds are identified as:

```
MAJOR.MINOR.PATCH-dev.YYYYMMDD.N
```

Where: - **YYYYMMDD**: Date in year-month-day format - **N**: Build number for that day starting at 1

Example: 1.2.0-dev.20230503.3

1.4 Version Storage and Access

Version information is stored in:

1. **BuildConfig**: For referencing during runtime
2. **Settings Screen**: Displayed to users for support purposes
3. **About Screen**: Complete version information (semantic version and build number)
4. **Crash Reports**: Automatically included in crash data
5. **API Requests**: Sent in the User-Agent header

1.5 Versioning of API and Components

1.5.a Backend API

The API follows its own versioning scheme: - Major API version in the URL path: `/api/v1/` - Minor API changes managed through the **API-Version** header

1.5.b Database Schema

Database schema versions are managed independently: - Incremented with each schema change - Migrations provided between consecutive versions - Tracked through Room's version number

1.6 Version Compatibility

1.6.a Minimum OS Version

The minimum OS version supported by each release is documented and enforced:

App Version	Minimum Android	Minimum iOS
1.0.0 - 1.1.0	8.0 (API 26)	14.0
1.2.0+	8.0 (API 26)	14.0
2.0.0+ (planned)	9.0 (API 28)	15.0

1.6.b API Compatibility

Each app version documents which API versions it supports:

App Version	Supported API Versions
1.0.0	v1.0
1.1.0 - 1.2.0	v1.0, v1.1
2.0.0+ (planned)	v2.0+

1.7 Version Bumping Procedures

1.7.a When to Bump Versions

- **MAJOR:** Significant UI redesigns, breaking API changes, major feature sets
- **MINOR:** New features, non-breaking enhancements, substantial improvements
- **PATCH:** Bug fixes, performance improvements, minor UI adjustments

1.7.b Version Bump Process

1. Version bumps are proposed during release planning
2. Tech Lead approves version changes
3. Version is updated in:
 - `build.gradle` for Android
 - `Info.plist` for iOS
 - Release documentation

1.8 Release Branches

Each version has its own Git branch:

- Feature development: `feature/feature-name`
- Release preparation: `release/1.2.0`
- Hotfixes for released versions: `hotfix/1.1.1`

1.9 Version in Documentation

All documentation must clearly indicate:

1. The app version(s) it applies to
2. Any version-specific considerations
3. Deprecated features and when they'll be removed

1.10 Archiving and Maintenance

Older versions are maintained according to this schedule:

- Current version: Full support
- Previous minor version: Security and critical bug fixes only
- Previous major version: Security fixes for 6 months
- Older versions: No support

1.11 Examples

1.11.a Standard Release Pattern Example

1. Development starts on version 1.2.0
2. Internal builds: 1.2.0-alpha.1, 1.2.0-alpha.2, etc.
3. Beta testing: 1.2.0-beta.1, 1.2.0-beta.2, etc.
4. Release candidates: 1.2.0-rc.1, 1.2.0-rc.2
5. Final release: 1.2.0
6. Bug fix: 1.2.1

1.11.b Hotfix Example

1. Critical bug discovered in production version 1.2.0
2. Hotfix branch created from tag 1.2.0
3. Fix developed and tested
4. Released as version 1.2.1
5. Fix merged back to development branch