ChatGPT

SanctissiMissa – Product Requirements Document (PRD)

Document Version: 1.0
Date: 18 June 2025
Prepared by: Product Requirements Document Writer (AI PM)

---

1. Executive Summary

SanctissiMissa is a free, web-first and cross-platform liturgical companion centred on the **1962 Roman Missal** and the **Traditional Roman Breviary**. It provides faithful, clergy, and liturgy enthusiasts with complete Latin/English texts, an advanced calendar engine, rich devotional content, and modern productivity aids such as reminders and voice journaling.
The project's guiding principles are **fidelity to tradition, accessibility on any device, and privacy-respecting personal growth tools**.

Business Outcomes

Goal	Metric	Target
Grow monthly active users	MAU (web + native)	50 k within 12 months
Improve daily prayer adherence	% users completing ≥3 canonical hours/day	25 % within 6 months
Voice-journal adoption	% active users creating ≥1 entry/week	20 % by month 9
System reliability	Crash-free sessions	≥ 99 %
Offline coverage	% liturgical texts available without network	100 % core texts

---

2. Background & Problem Statement

Traditional Latin-rite Catholics face a fragmented digital ecosystem—PDF missals, static websites, and platform-specific apps of varying quality. Content often lacks bilingual display, advanced calendrical logic, or modern user-experience features (offline, reminders, voice notes). A unified, perpetually free, and device-agnostic platform is needed.

---

3. Target Users & Personas

1. **Daily Devotee (Alice, 29, teacher)** – Prays parts of the Office on her phone; needs subtle reminders and quick text access offline.
2. **Choir Director (Br. Joseph, 42)** – Prepares chant for sung Mass; requires accurate propers, rubrics, and print-friendly output.
3. **Priest on the Road (Fr Miguel, 35)** – Celebrates Mass while travelling; needs full missal texts and rubrical guidance offline.
4. **Curious Newcomer (Liam, 21)** – Exploring the Extraordinary Form; benefits from side-by-side Latin/English and explanatory tool-tips.

---

4. Scope

4.1 In-Scope

- Full 1962 Missal (ordinary, propers, commons, rubrics)
- Complete 1962 Breviary with eight canonical hours
- Advanced liturgical calendar (temporal & sanctoral, precedence)
- Latin devotions (Rosary, Divine Mercy, Stations, Litanies, Novenas)
- Guided-prayer modes and bead visualisation
- Voice Journal with local-first storage and optional encrypted backup
- Reminder system for canonical hours
- Responsive PWA + React Native bundle (Android first)
- Offline capability via service-worker/SQLite/IndexedDB
- Accessibility (WCAG 2.2 AA), dark/sepia modes, adjustable fonts

4.2 Out-of-Scope (Phase 1)

- Community/social sharing features
- iOS App Store submission (TBD post-launch)
- Live-streamed Mass or external media integrations

---

5. User Stories (Sample)

1. **As a lay faithful**, I want a subtle banner 15 min before Vespers so that I can pray on time without disruptive notifications.
2. **As a priest**, I want to preload Mass propers for tomorrow so that I can celebrate even without data connectivity.
3. **As a user**, I want to record a 90-second reflection linked to today's Collect so that I can revisit it next liturgical year.
4. **As a commuter**, I want voice navigation so that I can advance through the Rosary hands-free.

---

6. Functional Requirements

6.1 Content & Calendar

- Calculate movable feasts based on Easter; support local calendars.
- Surface rubrical precedence for commemorations and rank clashes.
- Store bilingual text pairs; render side-by-side or toggle per user.

6.2 Navigation & UI

- Bottom-tab nav on phones; persistent sidebar on tablets/desktop.
- Adaptive split-view for Latin/English or text/commentary.
- Guided Prayer Mode: timed progression, bead tracker, TTS option.

6.3 Personalisation

- Theme: light / dark / sepia; font size slider; Latin-only, English-only, parallel.
- Bookmarks, favourites, last-read history per user.

6.4 Reminders & Notifications

- Customisable canonical-hour schedule; respect quiet hours.
- In-app banner + OS notification with deep-link to office.

6.5 Voice Journal

- Long-press on any text → Record Journal Entry.
- Audio capture (Opus), local encryption, optional cloud sync.
- Auto-transcription (on-device Whisper-tiny) for searchability.

6.6 Offline & Sync

- Service-worker caches static assets and upcoming 7 days of texts.
- IndexedDB/Web-SQL fallback; native uses SQLite.
- Background sync for calendar updates and cloud backups.

6.7 Search & Filtering

- Full-text search across Latin and English; fuzzy match; filters by feast rank, season, devotion type.

---

7. Non-Functional Requirements

Category	Requirement
Performance	≤ 2 s first meaningful paint on 3G; ≤ 100 ms nav latency
Reliability	99 % crash-free sessions; graceful offline degradation
Security	OWASP Mobile + PWA top-10 compliance; end-to-end AES-256 for cloud journal backups
Privacy	No third-party analytics; opt-in pseudonymous telemetry

Category	Requirement
Accessibility	WCAG 2.2 AA minimum; screen-reader friendly
Internationalisation	UI strings externalised; future vernacular translations

8. Technical Implementation Overview

8.1 Architecture (derived from *StAndroidMissal-ARCHITECTURE*)

- **Platform-agnostic core** (`/core`) with business logic, TS types, Redux store.
- **Services**: `CalendarService`, `TextService`, `DataManager`.
- **Adapters** (`/platforms/native`, `/platforms/web`) bridge storage, FS, device info.
- **UI Layer**: React Navigation (tabs + stack), reusable components, responsive hooks.
- **Data**: Pre-packaged liturgical JSON/SQL; migration pipelines for updates.
- **Build**: Vite for web; React Native CLI for Android.

8.2 Tech Stack

Layer	Technology
Language	TypeScript 5.x
Framework	React 18 + React Native 0.74
State	Redux Toolkit + RTK Query
Storage (web)	IndexedDB via Dexie
Storage (native)	`react-native-sqlite-storage`
Offline	Service Worker (Workbox)
Styling	Tailwind CSS / React Native Wind
Audio	MediaRecorder API (web); `react-native-audio-recorder-player`
Testing	Jest, React Testing Library, Detox (native e2e)
CI/CD	GitHub Actions, Expo Application Services (build only)

8.3 Leveraging Weak Agentic Coding LLMs

Task	LLM Role	Constraints
Code generation for boilerplate slices & hooks	Generate TS templates based on high-level spec	Limit to ≤400 token prompts; human review mandatory
Unit-test scaffolding	Autofill Jest test skeletons	No network access; must import local modules only
Data import scripts	Convert CSV → SQLite statements	Agent must output deterministic scripts; checksum verified

Task	LLM Role	Constraints
Internal docs	Summarise TS interfaces to MD	Output checked into docs folder; CI fails on hallucination via schema-lint

Guard-Rails: LLM containers run with read-only repo access, isolated from secrets. Outputs go through static analysis + ESLint before merge.

8.4 Deployment

- **Web**: GitHub Pages + Cloudflare cache; automatic preview per PR.
- **Android**: Play Store (internal testing) → production track.
- **Versioning**: Semantic Ver (CalVer fallback).
- **Rollback**: Previous PWA bundle kept; Android staged rollout.

---

9. Implementation Roadmap (SOTA 48-Hour Benchmark — zero padding)

Principle: Assume state-of-the-art, fully autonomous coding LLMs working around the clock with instant CI/CD feedback loops. Durations below are *pure execution windows* with no baked-in buffer.

Stream	Window (Hours)	Ideal Duration	Key Deliverables	Dependence
A. Prompt Engineering & Repo Bootstrap	0 – 3	**3 h**	Ontology, guard-rails, mono-repo scaffold, CI pipeline	–
B. Core Logic Services	3 – 12	**9 h**	CalendarService, TextService, schema validators	A (partial)
C. Storage & Offline Layer	6 – 15	**9 h**	IndexedDB/SQLite adapters, Workbox SW	overlaps B
D. UI Assembly & Navigation	9 – 24	**15 h**	Responsive screens, navigation shell, theming system	overlaps B+C
E. Voice Journal & Reminders	15 – 30	**15 h**	Audio stack, Whisper-tiny integration, notification engine	B+C ready
F. Integrated Testing & Hardening	24 – 42	**18 h**	Auto-generated test suite, perf budget, accessibility audit	continuous
G. Beta Cut & Production Launch	42 – 48	**6 h**	PWA deploy, Play Store rollout, smoke tests	after critical F passes

Total elapsed wall-clock time: 48 hours (2 calendar days).

10. Risks & Mitigations

Risk	Impact	Mitigation
Complex precedence rules produce wrong propers	Medium	Extensive unit tests with historical calendars
Browser storage quota limits large audio files	High	Compress to 48 kbps Opus; prompt user to offload to cloud
LLM hallucination in code generation	Medium	Static analysis + human code review gate
Changing ecclesiastical regulations (*Traditionis Custodes*)	Low–Med	Config-driven permissions; abstract text sources

11. Appendix

- Source documents: Architecture filecite turn0file3 , Checklist filecite turn0file4 , Project Overview filecite turn0file2 , Web-First Analysis filecite turn0file7 , Reminders filecite turn0file1 , Latin Prayers Nav filecite turn0file0 , Voice Journal filecite turn0file6 .
- Glossary: EF = Extraordinary Form; PWA = Progressive Web App; LLM = Large Language Model.

End of Document