Offline-First Translation PWA

Technische Architektur für eine vollständig offline-fähige Übersetzungs-App mit nativen App-Wrappern.

Zur Implementierung | **Einstellungen**

Für Ulrich

Warum nicht nur native Apps? Weil wir mit einer Web-App iOS *und* Android aus einer Codebasis bedienen. Das spart 60% Entwicklungszeit. Offline-Übersetzung funktioniert genauso gut wie bei nativen Apps.

Aktueller Tech-Stack

Produktions-Services

Translation

Google Cloud → MyMemory → LibreTranslate

Dreistufige Fallback-Kette für maximale Verfügbarkeit.

Text-to-Speech

Google Cloud Neural2/Chirp3HD mit Browser SpeechSynthesis Fallback

HD-Qualität, 22 Sprachen unterstützt.

Speech-to-Text

Web Speech API (Chrome/Edge)

Netzwerkabhängig, nur in Chromium-Browsern.

PWA + Sync

vite-plugin-pwa mit Workbox

Live-Sessions via Supabase Realtime.

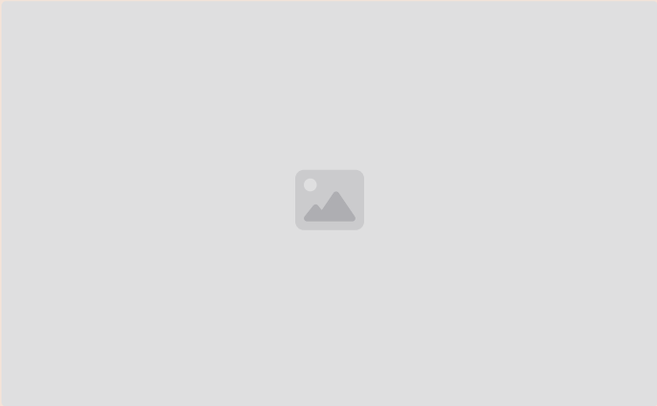
Für Ulrich

Aktuell funktioniert die App nur mit Internet. Wie WhatsApp Web: Super, wenn WLAN da ist. Aber im Hafen ohne Netz? Nutzlos.

Native Apps haben denselben Problem, wenn wir nicht Offline-Modelle einbauen.

Offline-Technologie-Entscheidungen

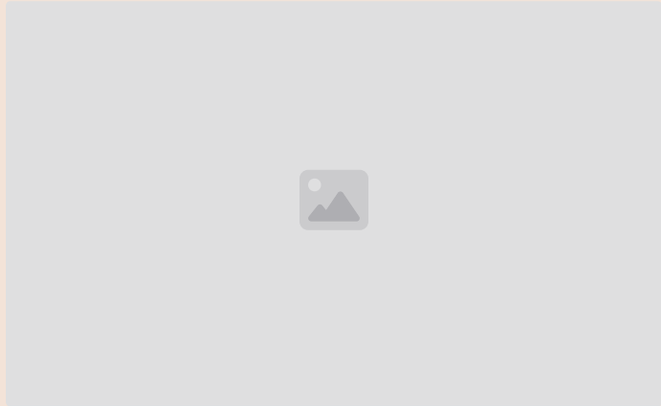
Ausgewählte Technologien nach umfassender Research-Phase. Alle Komponenten WebAssembly-basiert für maximale Performance.



Translation Engine

@huggingface/transformers + Opus-MT ONNX

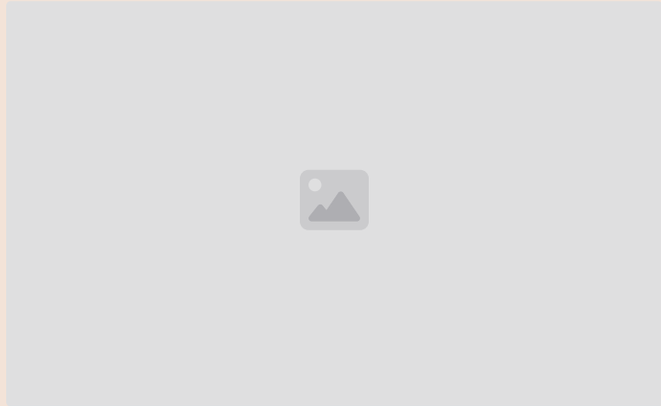
35MB pro Sprachpaar. Quantisiert auf int8. 22 Hauptsprachen abgedeckt.



TTS Offline

Sherpa-ONNX WASM (Matcha)

10MB pro Stimme. Natürliche Aussprache. Web Speech API als Fallback.



STT Streaming

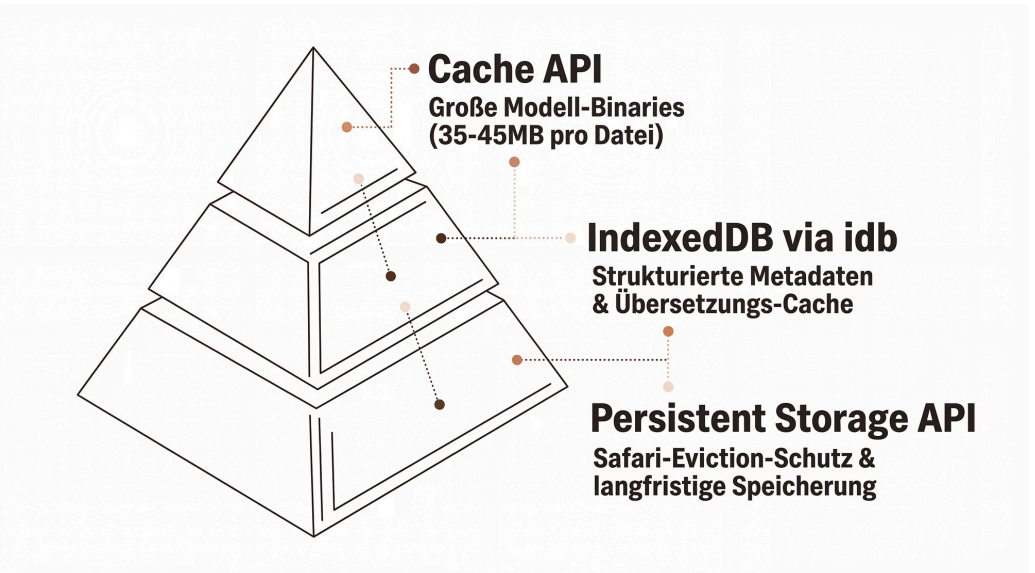
Sherpa-ONNX Zipformer

45MB Modell. Streaming-fähig. 20x schneller als Echtzeit.

Für Ulrich

Wir laden Mini-KI-Modelle runter (wie Spotify Songs). Danach funktioniert alles ohne Internet. Eine native App würde *exakt dieselben* Modelle nutzen.

Storage-Architektur



Speicher-Management

Cache API: ONNX-Modelle, WASM-Binaries

IndexedDB: Translation Cache, TTS Audio, Metadaten

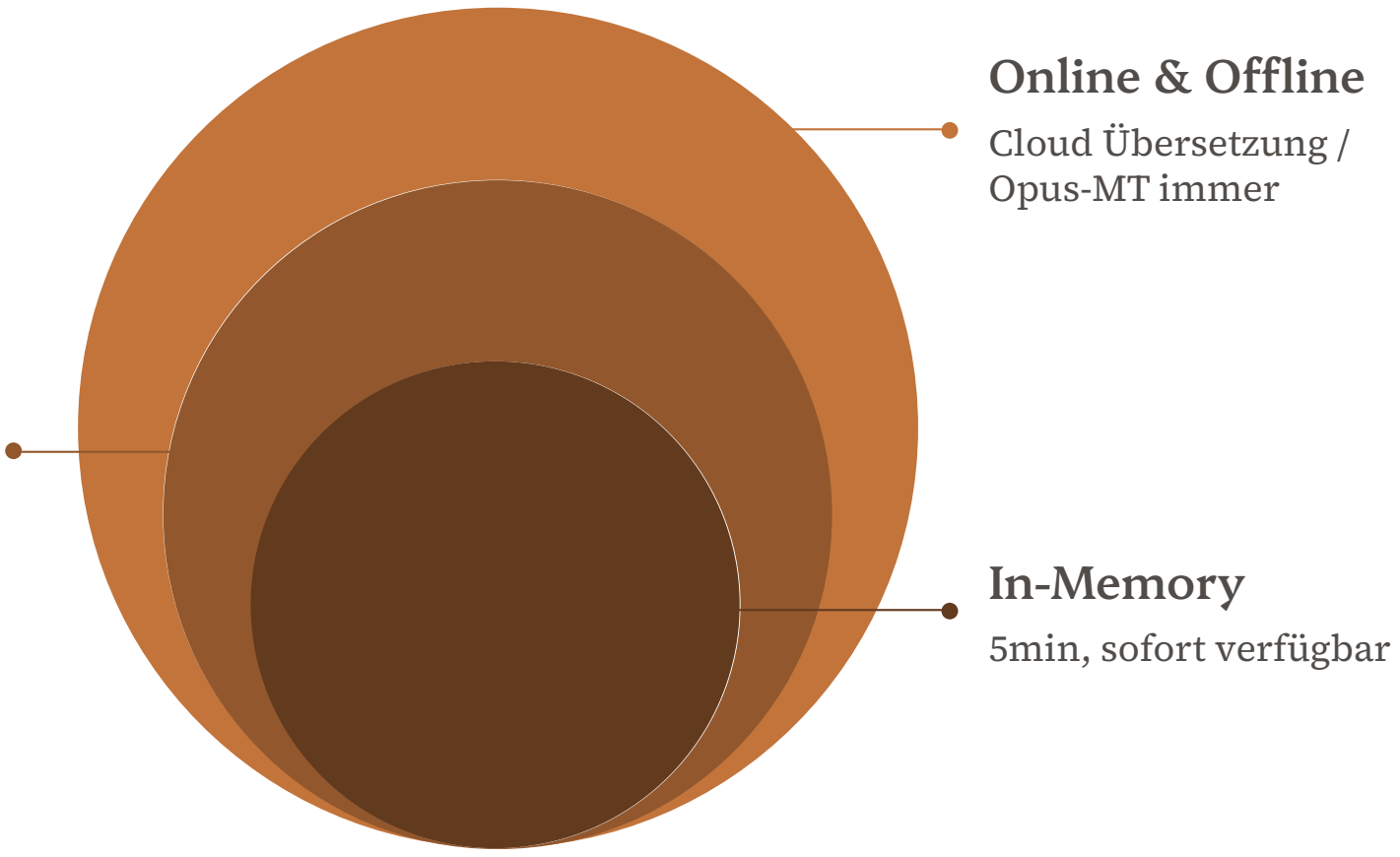
Persistent Storage: Safari-Schutz vor automatischer Löschung

- requestPersistentStorage() beim App-Start
- Quota-Monitoring mit Storage Estimates
- iOS "Zum Homescreen"-Prompt bei Safari

Für Ulrich

Dein iPhone löscht manchmal App-Daten, wenn Speicher knapp wird. Wir bitten iOS: "Bitte nicht löschen!" Funktioniert nur, wenn die App auf dem Homescreen ist.

Schritt 1–2: Persistent Translation Cache



Intelligente Caching-Strategie verkürzt Ladezeiten und ermöglicht echte Offline-Nutzung.

IndexedDB Schema (idb)

```
interface TranslationCacheEntry {
  key: string // "de|en|hallo welt"
  translatedText: string
  provider: string
  timestamp: number
}

interface ModelMeta {
  id: string // "opus-mt-de-en"
  type: 'translation' | 'tts' | 'stt'
  sizeBytes: number
  version: string
}
```

Neue Dateien

- `src/lib/offline/db.ts` — Schema Definitionen
- `src/lib/offline/storage-manager.ts` — Quota Management
- `src/lib/offline/translation-cache.ts` — Cache Logic

Änderungen

`src/lib/translate.ts` Zeile 182-186: IndexedDB Check nach In-Memory

`package.json`: `idb` Package (~3KB)

Für Ulrich

Stell dir vor: Du übersetzt "Wo ist der Hafen?" in Neapel. Wir merken uns die Übersetzung für 30 Tage. Nächste Woche in Barcelona? Sofort da, kein Internet nötig.

Schritt 3: Network Status Management

1

Network Detection

`navigator.onLine` Events + 10s Heartbeat-Ping

2

React Context

App-weiter State: Online/Degraded/Offline

3

Engine Selection

Automatische Provider-Wahl je nach Status

NetworkMode-Typen

```
type NetworkMode =
  | 'online'    // Volle Cloud-Power
  | 'degraded'  // Langsam, Fallbacks
  | 'offline'   // Nur lokale Engines
```

Heartbeat prüft alle 10 Sekunden mit kleinem fetch zu bekanntem Endpoint.

useNetworkStatus Hook

```
const {
  mode,
  isOnline,
  translationEngine,
  ttsEngine,
  sttEngine
} = useNetworkStatus()
```

Automatische Engine-Selektion basierend auf Verfügbarkeit.

Für Ulrich

Die App checkt ständig: "Hab ich Internet?" Wenn ja: beste Cloud-Qualität. Wenn nein: lokale Mini-KI übernimmt. Du merkst den Unterschied kaum.

Schritt 4–5: Offline Translation Engine

Opus-MT ONNX Modelle mit Pivot-Strategie für 22 Sprachen. Quantisiert auf int8 für mobile Performance.

Modell-Strategie

Direkte Paare

DE↔EN, EN↔FR/ES/IT/PT/NL/RU/PL/CS

~35MB pro Richtung

Pivot via Englisch

DE→IT = DE→EN + EN→IT

Zweistufig für seltene Kombinationen

Verfügbare Modelle

- Xenova/opus-mt-de-en, en-de
- Xenova/opus-mt-en-{fr,es,it,pt,nl,ru,pl,cs}
- Rückrichtungen: fr-en, es-en, etc.

Model Manager API

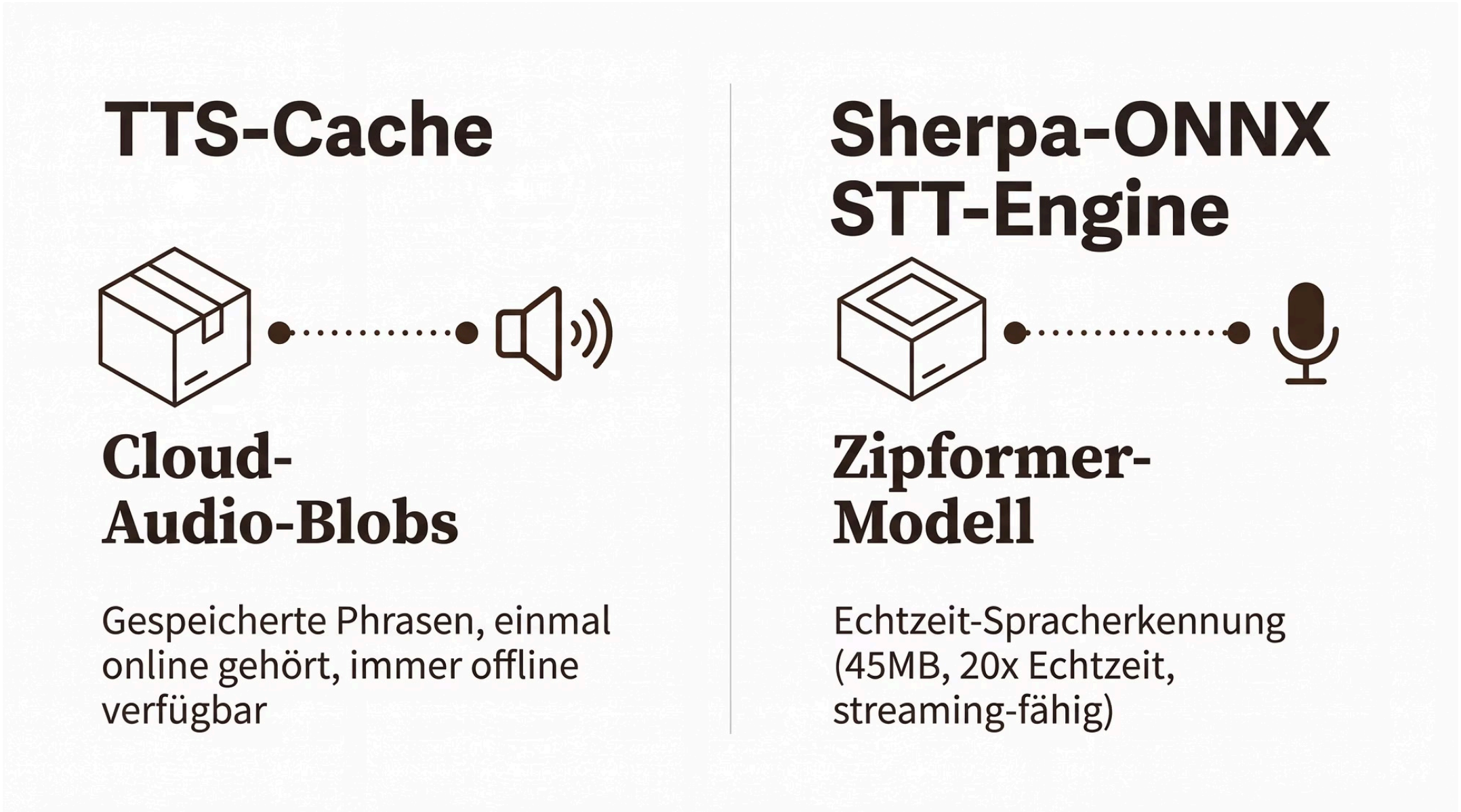
```
// Model Download
await downloadModel(
  'opus-mt-de-en',
  (progress) => console.log(progress)
)

// Prüfen
const downloaded =
  await isModelDownloaded('opus-mt-de-en')

// Übersetzen
const result = await translateOffline(
  'Hallo Welt',
  'de',
  'en'
) // → "Hello World"
```

Cache API Storage: Modelle als Blobs, nicht in IndexedDB (performanter bei großen Dateien).

Für Ulrich

22 Sprachen = 22 kleine Übersetzer-Gehirne. Du lädst nur die runter, die du brauchst. Mittelmeer-Kreuzfahrt? Italienisch + Griechisch = 70MB. Passt locker.

Schritt 6–7: TTS & STT Offline



TTS Audio Cache

Strategie: Cloud-Qualität persistent cachen

- VOR Cloud-Call: IndexedDB Check
- NACH Cloud-Call: Blob cachen (30 Tage TTL)
- Phrasen-Preload für häufige Sätze

src/lib/offline/tts-cache.ts

```
getCachedTTSAudio(text, lang, quality)
cacheTTSAudio(text, lang, quality, blob)
preloadPhraseAudio(phrases[])
```

Sherpa-ONNX STT

Zipformer Modell: 45MB, streaming-capable

- AudioContext + ScriptProcessorNode
- 20x schneller als Echtzeit
- Fallback-Kette: Apple SpeechAnalyzer → Web Speech → Sherpa

src/lib/offline/stt-engine.ts

```
createSherpaSTTEngine(): STTEngine
// start(), stop(), cleanup
// provider: 'sherpa-onnx'
```

Für Ulrich

TTS: Einmal online "Guten Morgen" angehört? Wird gespeichert. Offline immer noch dieselbe Qualität.

STT: Du sprichst ins Mikrofon, Mini-KI versteht dich. Ohne Internet. Wie Siri, aber lokal.

Schritt 8–9: Settings UI & Offline Indicator

Settings Page Features

1

Offline-Sprachen

Download/Löschen pro Sprachpaar mit Progress-Bar

2

Spracheingabe

Sherpa STT Modell (45MB) Download-Button

3

Speicher-Management

Gesamtverbrauch, Cache leeren, Persistent Storage Status

4

Stimm-Qualität

HD/SD Toggle (von TranslationPanel hierhin verschoben)

Header Offline Indicator

🟢 Online

Cloud-Provider aktiv

Beste Qualität

🟠 Offline

Lokale Engines

Cache + Modelle

Klick öffnet Tooltip mit Details: Welcher Provider, welche Engine, verfügbare Sprachen.

`Header.tsx` Zeile 70+: Settings-Icon (⚙️) + Status-Dot integriert.

Neue Components: LanguagePackCard, StorageIndicator

Für Ulrich

Oben rechts siehst du einen Punkt: Grün = Internet da. Orange = Offline-Modus. In Einstellungen lädst du Sprachen runter wie Netflix-Serien.

Schritt 10–11: Service Worker & Phrase Packs

Workbox Config Erweiterungen

`vite.config.ts` angepasst:

- **globPatterns:** `*.onnx`, `*.wasm` inkludiert
- **offline-models:** CacheFirst, 90 Tage
- **offline-tts-audio:** CacheFirst, 30 Tage
- **Google Translation API:** StaleWhileRevalidate
- **navigateFallback:** `/index.html` für SPA-Routing

Cruise Phrase Packs

Vorgefertigte Phrasen-Sammlungen für typische Kreuzfahrt-Szenarien:

Hafen & Transport

Sehenswürdigkeiten

Essen & Trinken

Shopping

Notfälle

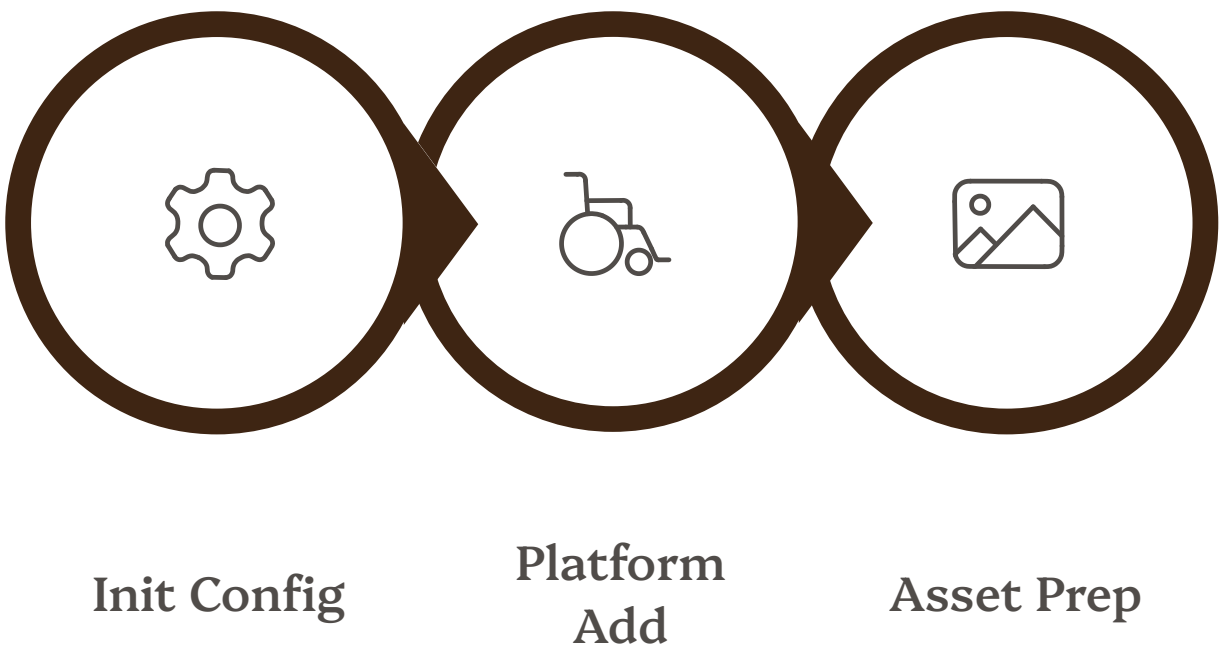
`phrases-mediterranean.json` (500 Phrasen), `phrases-common.json` (200 Basis)

Für Ulrich

Service Worker = unsichtbarer Helfer, der Dateien clever speichert. Phrase Packs = fertige Sätze wie "Wo ist die Toilette?" in 22 Sprachen vorinstalliert.

Schritt 12: Capacitor App Store Wrapper

Native iOS und Android Apps aus einer Codebase. Identische Offline-Funktionalität wie PWA, aber im App Store verfügbar.



Setup-Befehle

```
npm install @capacitor/core @capacitor/cli

npx cap init "GuideTranslator" \
  "com.fintutto.translator"

npm install @capacitor/ios @capacitor/android

npx cap add ios
npx cap add android

npx cap sync
npx cap open ios
npx cap open android
```

Vorteile vs. Pure PWA

App Store Präsenz

Auffindbarkeit, Reviews, Trust

Native APIs

Apple SpeechAnalyzer, bessere Permissions

Kein Safari-Eviction

iOS löscht keine App-Daten

Asset-Requirements: 1024x1024 Master-Icon → automatische Größengenerierung

Für Ulrich

Capacitor = Verpackungsmaschine. Wir schreiben *einmal* Code, Capacitor macht daraus eine iOS-App *und* eine Android-App. Wie ein Übersetzer für App Stores. Eine native App pro System zu bauen würde 3x länger dauern.

Implementierungs-Roadmap & Verifizierung

1

Phase 1: Fundament (Woche 1-2)

Schritte #1-3: IndexedDB, Storage, Network Context

2

Phase 2: Offline Engines (Woche 3-4)

Schritte #4-7: Translation, TTS Cache, STT

3

Phase 3: UI & UX (Woche 5)

Schritte #8-9: Settings, Indicator, Downloads

4

Phase 4: PWA & Wrapper (Woche 6)

Schritte #10-12: Service Worker, Phrases, Capacitor

Test-Checkliste

- **Offline Translation:** DevTools Network disconnect → Text übersetzen → Opus-MT funktioniert
- **Persistent Cache:** Übersetzen → App schließen → Reopen → sofortiger Cache-Hit
- **TTS Cache:** Online vorlesen → offline → gleicher Text → gecachte Cloud-Qualität
- **STT Offline:** Sherpa laden → disconnect → Spracheingabe → Erkennung läuft
- **Network Indicator:** Disconnect → Header "🟠 Offline" → Reconnect → "🟢 Online"
- **Settings:** Sprachpaar download/delete → Storage-Anzeige update
- **PWA Install:** Chrome "App installieren" → Standalone → Offline-funktional
- **Safari 7-Tage-Test:** iOS "Zum Homescreen" → 7+ Tage warten → Modelle noch vorhanden

Für Ulrich — Finale Antwort

Warum nicht nur native Apps? Weil wir mit dieser Architektur:

- iOS + Android aus *einer* Codebasis bauen (60% Zeitersparnis)
- Dieselben Offline-Modelle nutzen wie native Apps
- Zusätzlich als PWA verfügbar sind (Web-Zugang ohne Install)
- Schneller Updates deployen können (kein App Store Review für Web)

Zwei native Apps separat = 6 Monate. Diese Lösung = 6 Wochen. Gleiche Qualität.