TamilTheni Architecture Document

Version: 2.0
Last Updated: January 2026
Author: Peoria Tamil School Development Team

Table of Contents

Executive Summary

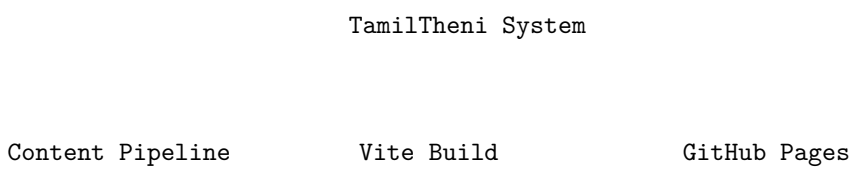
TamilTheni is a Tamil language learning web application designed for the FETNA Tamil Theni Competition. The application is a static single-page application (SPA) hosted on GitHub Pages, featuring five distinct learning modules targeting different Tamil language skills including vocabulary, sentence construction, translation, and word discovery.

Key Architectural Decisions

Decision	Rationale
Vite	Modern, fast build tool with instant HMR and optimized production builds
TypeScript	Static typing for better maintainability and error catching
JSON Data	Structured, interoperable data format separated from logic
CSS Modules	Component-scoped styling (via standard CSS imports)
Python Tooling	Offline data processing pipeline for content generation

---

System Overview



```
(Python Scripts)          (TS -> JS Bundle)          (Deployment)


JSON Data Files           External APIs
(src/data/*.json)         (Wikipedia, AI)
```

Architecture Diagram

Application Layer

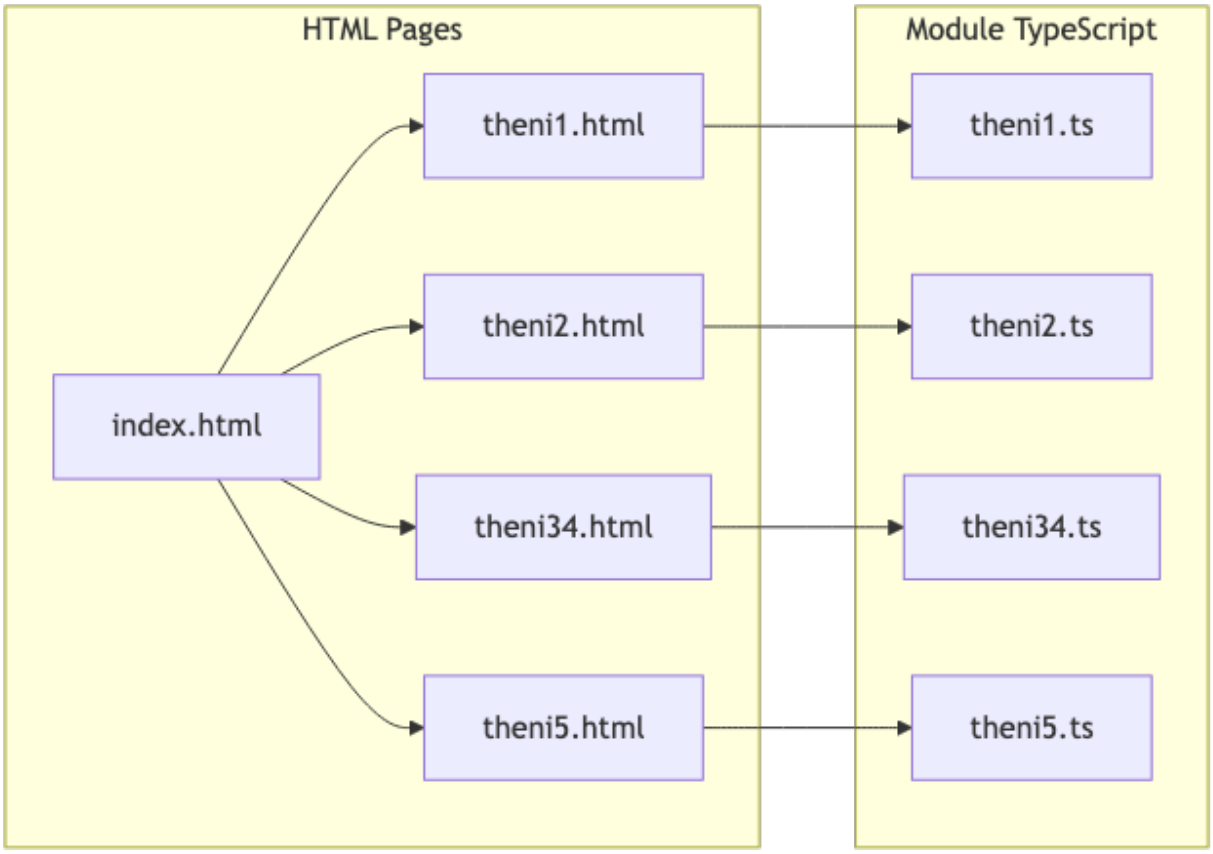


Figure 1: Application Layer

Shared Infrastructure

Data Flow

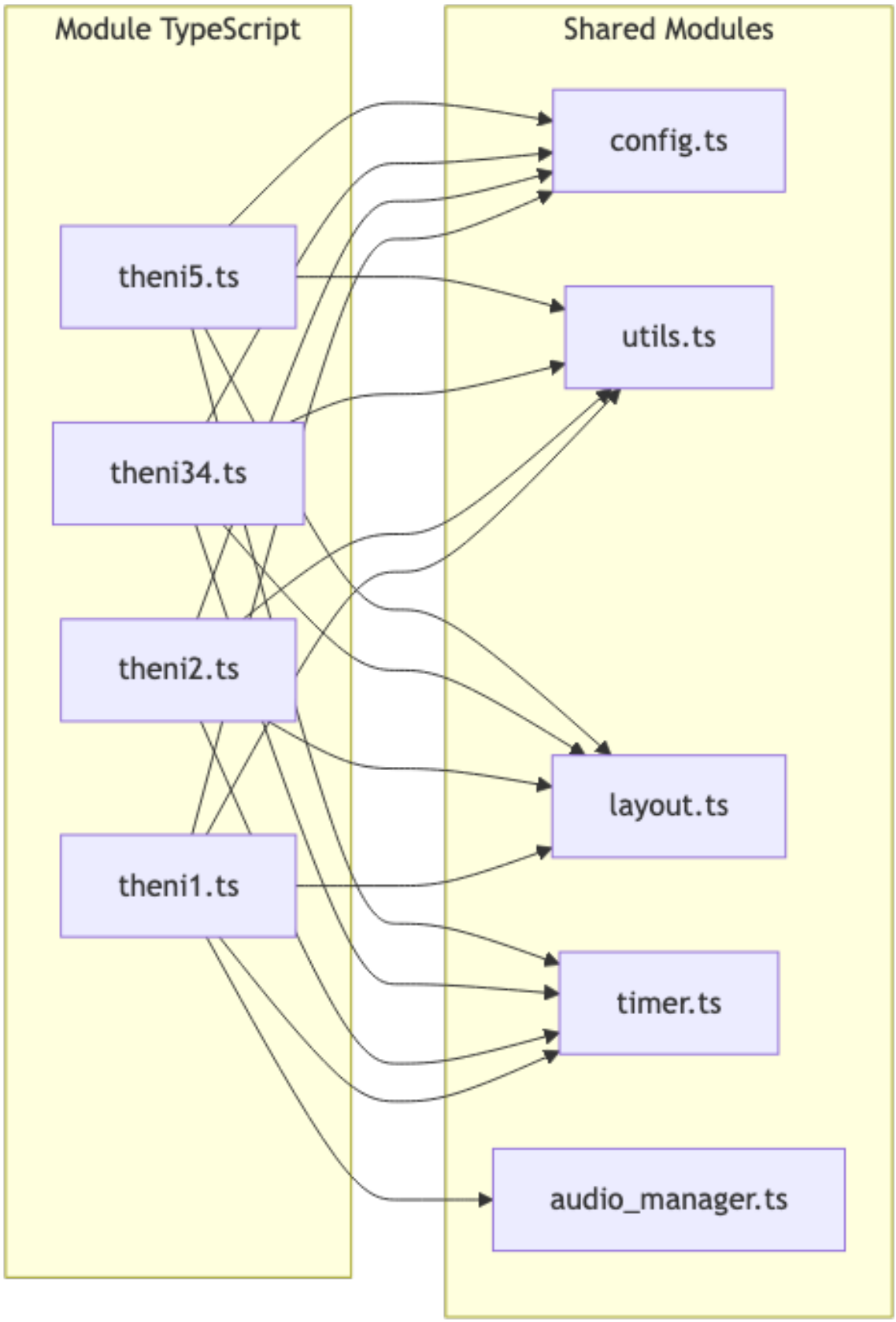


Figure 2: Shared Infrastructure

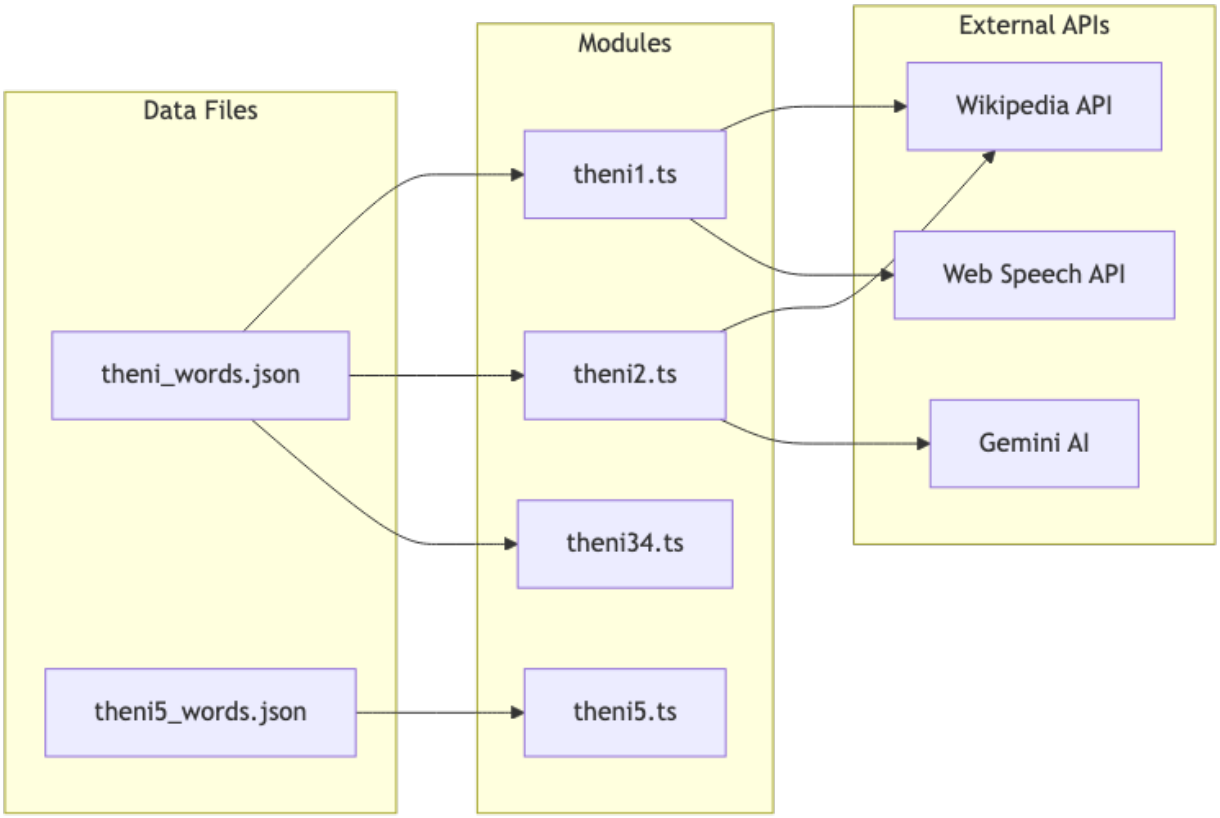


Figure 3: Data Flow

Frontend Architecture

Directory Structure

```
tamiltheni/
  public/                 # Static assets (images, fonts) served efficiently
  src/                    # Source code
      css/                # Stylesheets modularized by page
      js/                 # TypeScript logic files
      data/               # JSON data files (Single Source of Truth)
      types/              # TypeScript interface definitions
  html/                   # HTML entry points for each module
  test/                   # Test files
      bat/                # Build Acceptance Tests
      unit/               # Unit tests
  documentation/          # Project documentation
      ARCHITECTURE.md     # This file
      REQUIREMENTS.md     # Product requirements
  index.html              # Main entry point
  docs/                   # Production build output (GitHub Pages root)
```

TypeScript Strategy

We use TypeScript to enforce data contracts and reduce runtime errors. Key interfaces include:

```typescript
// Word Data Structure
interface Word {
    id: number;
    category: string;
    word_en: string;
    word_ta: string;
    difficulty: "D1" | "D2";
    // ...other fields
}
```

The build process (`tsc && vite build`) transpiles this to optimized JavaScript bundles.

Shared Modules

Module	Purpose	Key Exports
`config.ts`	Centralized configuration	`config` object
`utils.ts`	Utility functions	`Utils` class
`layout.ts`	UI component injection	`Layout` class
`timer.ts`	Countdown timer engine	`Timer` class
`audio_manager.ts`	Text-to-Speech wrapper	`AudioManager` class

Data Architecture

JSON Data Files (`src/data/`)

Data is stored in standard JSON format, allowing easy manipulation by Python scripts and straightforward import by TypeScript.

- `theni_words.json`: Contains the main dataset of ~800 words.
- `theni5_words.json`: Contains the clue-based dataset for Theni 5.

Word Data Schema (`theni_words.json`)

```
[
  {
    "id": 1,
    "category": "Body Parts",
    "category_ta": "       ",
    "difficulty": "D1",
    "word_en": "ear",
    "word_ta": "  ",
    "image_word": "ear",
    "sentence_en": "I have an <b>ear</b> infection.",
    "sentence_ta": "              .",
    "complexity": 2
  }
]
```

Image Storage Strategy

- **Location:** `public/assets/images/theni12/`
- **Naming:** `{word_en}.jpg` (e.g., `ear.jpg`)
- **Fallback:** Wikipedia API fetch if local image missing

Python Tooling Pipeline

Pipeline Overview

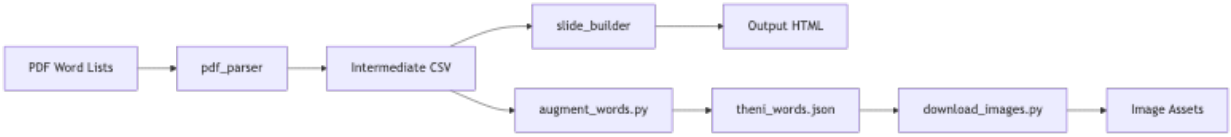


Figure 4: Python Tooling Pipeline

Deployment Architecture

GitHub Pages Configuration

```
# Deployment: docs/ folder on publish branch
Source: docs/
Branch: publish (primary)
```

Build Process

1. **Development**: `npm run dev` serves files from memory with hot replacement.
2. **Production**: `npm run build` runs `tsc` (type check) then `vite build`.
3. **Artifacts**: Minified JS/CSS and assets are output to `docs/`.

Module Deep Dives

Timer Module (`timer.ts`)

Configurable countdown timer with visual pie-chart representation and audio feedback.

Layout Module (`layout.ts`)

Injects common UI elements (headers, navigation, sidebars) into each HTML page at runtime, ensuring consistency.

Security Considerations

API Key Management

The Theni 2 module uses the Gemini AI API. Keys are stored in `localStorage` by the user.

> [!WARNING] Client-side API key storage is inherently insecure.

Future Considerations

1. **PWA Enhancements**: Improve offline capabilities.
2. **Backend**: Optional backend for user progress tracking.
3. **Testing**: Add unit tests (Vitest) and E2E tests (Playwright).

This document is maintained alongside the codebase.