ChatGPT

Final Production Structure for Ta'amimFlow

This document describes a proposed **production-ready** file structure for the Ta'amimFlow project, together with an import graph and a refactoring plan for integrating the cantillation extraction (Milestone 9/9.1) and audio engine (Milestone 10/10.2) into the existing application.

The goal of this restructuring is to turn the prototype modules delivered in Milestones 9–10 into a clean, maintainable product. It eliminates redundancy (no duplicate modules such as `milestone9.py` / `milestone9_plus.py`), clearly separates concerns, and preserves compatibility with the existing GUI and connectors.

1. Proposed Directory Layout

The existing `taamimflow` repository consists of a GUI written in PyQt6, connectors for retrieving Tanach text (local files or Sefaria API), and some utility code. The new cantillation and audio code should live in dedicated subpackages under `taamimflow/` to avoid namespace clutter and to keep the API surface clear:

```
taamimflow/
├── main.py                  # existing application entry point (GUI)
├── config.py                # configuration loader (existing)
├── connectors/              # text connectors (local_tanach, sefaria) –
unchanged
├── data/                    # XML definition files (tropedef.xml,
tropenames.xml, sedrot.xml)
├── gui/                     # PyQt6 GUI components
├── core/                    # NEW: core logic for cantillation, context and
matching
│   ├── __init__.py
│   ├── cantillation.py      # **Milestone 9/9.1** functionality: text
normalisation,
│   │                        # trope extraction, group mapping, context
flags,
│   │                        # decision tree matching, debug explanations
│   ├── decision_tree.py     # builds a decision tree from tropedef.xml
│   ├── fsm_phrase_logic.py  # finite-state machine for phrase context flags
(verse/chapter/aliyah)
│   ├── aliyah_parser.py     # parser for sedrot.xml to determine aliyot
│   └── timing_map.py        # computes timing maps for karaoke highlighting
├── audio/                   # NEW: audio synthesis engines
│   ├── __init__.py
│   ├── audio_engine.py      # **Milestone 10 MVP** – sinus-wave synthesiser
│   ├── concat_audio.py      # **Milestone 10.2** – real concatenative
synthesis with cross-fades
│   ├── tradition_profiles.py# (optional) metadata describing available
```

```
cantillation styles
│   └── utils.py            # helper functions (loading samples, scaling
volume, etc.)
├── utils/                  # existing helper modules (paths.py, refs.py,
etc.)
└── tests/                  # test suite (Milestone 9/10 should be covered
by new tests)
```

Data files

The `data/` folder holds the XML definitions and will remain unchanged:

```
taamimflow/data/
├── tropedef.xml            # melodic definitions for each trope (many
traditions)
├── tropenames.xml          # mapping of trope codes to human-readable names
├── sedrot.xml              # book/chapter/verse ranges for parashiot and
aliyot
├── custom_sedrot.xml       # user-provided overrides (optional)
└── …
```

2. Import Graph

The import relationships between these modules should be as simple as possible. A directed acyclic graph (DAG) for the core and audio packages is shown below. Dependencies point from higher-level modules to lower-level ones:

```
main.py
└── gui.main_window (existing)                    # calls into connectors
and core
    ├── connectors.local_tanach / sefaria
    ├── core.cantillation
    ├── audio.audio_engine OR audio.concat_audio
    └── utils.paths

core.cantillation
├── core.decision_tree
├── core.fsm_phrase_logic
├── core.aliyah_parser
├── core.timing_map        # only if karaoke/timing features are enabled
└── taamimflow.data (via utils.paths.find_data_file)

audio.audio_engine
├── pydub.AudioSegment (external dependency)
└── optionally tradition_profiles

audio.concat_audio
├── pydub.AudioSegment
```

```
├── audio.utils
└── tradition_profiles

audio.tradition_profiles (optional)
└── loads definitions from data/tropedef.xml or separate JSON

fsm_phrase_logic
└── aliyah_parser     # uses sedrot.xml

decision_tree
└── taamimflow.data (tropedef.xml)
```

Rationale

- `main.py` **and the GUI** should not import low-level modules directly. Instead, they call into the high-level APIs provided by `core.cantillation` and `audio.*`. This ensures that the GUI remains decoupled from the internal logic and can be swapped or updated independently.

- `core.cantillation` orchestrates the entire cantillation extraction pipeline. It imports the decision tree and FSM modules to perform context matching and context flag computation. The GUI never needs to know about the internal decision tree or FSM.

- `audio.audio_engine` is the MVP audio synthesiser (sinus oscillator). It can be replaced by `audio.concat_audio` when concatenative synthesis is available. Both implement a common interface, such as `play_note_sequence(notes, tempo=120, volume=0.8)`, so the caller does not need to change code.

- `tradition_profiles` provides metadata describing which melodic profiles are available (Ashkenazi, Sephardi, etc.) and where the associated audio samples live. This allows the concatenation engine to load the correct samples.

3. Integration and Refactoring Plan

Below is a step-by-step plan for integrating these new modules into the existing codebase. Because the GitHub API is read-only for this session, you will need to perform these changes manually in your local clone of the repository.

1. **Create the new subpackages** `core/` and `audio/` under `taamimflow/`. Add `__init__.py` files to each to mark them as packages.

2. **Add** `cantillation.py`: copy the contents of your improved cantillation extraction module (formerly `milestone9_plus.py`) into `taamimflow/core/cantillation.py`. Update imports at the top to use relative imports (e.g., `from .decision_tree import DecisionTreeMatcher`).

3. **Add support modules** to `taamimflow/core/`: copy `decision_tree.py`, `fsm_phrase_logic.py`, `aliyah_parser.py`, and `timing_map.py` into this folder. Ensure they import `utils.paths.find_data_file` to locate XML files instead of hard-coded paths.

4. **Add** `audio_engine.py` **and** `concat_audio.py` to `taamimflow/audio/`. Both should implement a common interface: `synthesise(notes: List[Note], tempo: float, volume: float) -> AudioSegment` and `play(audio_segment)`. Use `pydub` for MP3/WAV output and crossfading, as indicated in the milestone analysis.

5. **Implement** `tradition_profiles.py` (optional): define data structures mapping tradition names (e.g. "Ashkenazi – Binder", "Sephardi – Syrian") to directories of pre-recorded trope segments. This allows the concatenation engine to locate the correct audio clips. A simple JSON or Python dictionary is sufficient for now.

6. **Remove redundancy**: delete the old `milestone9.py` and `milestone9_plus.py` files from the project once `cantillation.py` is in place. Their functionality is fully covered by the new `core` modules.

7. **Update the GUI** to use the new API:

8. Import `core.cantillation.TokenFull` (or whichever name you use) and call `cantillation.extract_tokens_with_notes(text, style=...)` instead of the old `tokenise` / `trope_parser` functions.
9. When audio is enabled in the config (`audio.enabled=true`), call `audio.audio_engine.synthesise` or `audio.concat_audio.synthesise` depending on the selected tradition and engine mode. The returned `AudioSegment` can be played with `pydub.playback.play()` or exported to a file.

10. Provide user-selectable options for tradition, tempo, and speed as indicated in the configuration guide [1].

11. **Integrate context flags**: Use `core.fsm_phrase_logic` to compute additional flags such as `start_of_chapter`, `end_of_aliyah`, etc., if these are needed for advanced musical variations. The flags can be included in the `TokenFull.context_flags` property and passed to the decision tree matcher.

12. **Tests**: Write a test suite under `tests/` to cover the new extraction pipeline and audio engine. Use sample verses from `tanach_data` and assert that the correct sequences of `Note` objects and audio durations are produced.

13. **Documentation**: Update the existing `docs/` folder to include:

    - **Extraction Pipeline**: a new page explaining the steps of the cantillation pipeline, including NFD normalisation, mark extraction, group mapping, context matching, and caching.
    - **Audio Engine**: instructions for installing dependencies (pydub, ffmpeg), selecting a tradition, and configuring audio options such as `default_volume`, `default_speed`, and `audio_format` [1].
    - **New Config Options**: mention any additional configuration keys required by the new modules (e.g., `tradition` selection, `audio_engine` choice). Existing options such as `audio.enabled` and `audio.tradition` are already defined in the config guide [1] and can be reused.

4. Notes on Dependencies and Performance

- **Dependencies**: the `pydub` library is used for audio synthesis and requires `ffmpeg` to be installed on the system. For the concatenation engine, you may also use `mutagen` to read MP3 metadata, and optionally `numpy` for waveform manipulation. These should be added to `requirements.txt`.

- **Performance**: the decision tree reduces the complexity of context matching from O(N) per note to O(1) by pre-compiling the conditions into a tree. Caching results (already implemented in your module) further speeds up repeated lookups. The concatenation engine loads audio samples lazily and caches them for reuse.

- **Extensibility**: new traditions or melodic styles can be added simply by appending additional `TROPEDEF` blocks to `tropedef.xml` or by adding new directories of audio samples and updating `tradition_profiles.py`.

5. Conclusion

This final production structure sets the foundation for a robust, modular cantillation system that goes beyond the Milestone prototypes. By organising the code into clear packages (`core` for logic and `audio` for synthesis), aligning with the existing configuration system, and providing a clean import graph, you ensure that future development—such as full Aliyah support, karaoke timing, or advanced phrase logic—can proceed without breaking the GUI or connectors. The refactoring plan outlined above can be applied incrementally in your local repository to transition from the current milestone branches to a cohesive product.

---

[1] CONFIG_GUIDE.md
https://github.com/Moriahise/taamimflow_project/blob/279375f5a7e7ccdfcd4fbc615c66355fded94a03/taamimflow/docs/CONFIG_GUIDE.md