Member responsibilities (equal split)

Ahmed — Project Lead & Assembler / Orchestrator (owner: `assembler`, `orchestration`, CI glue)

Files / folders to own

- `src/pipeline/assembler.py` — implement `assemble(frames_dir, audio_file, out_path)` that generates an MP4.
- `src/api.py` — simple CLI/REST entrypoint: `ttv_run(script_path, out_path)`.
- `src/cli.py` (optional) — `python -m ttv run --script data/sample_scripts/demo.txt`.
- Integration tests in `tests/test_end_to_end.py`.
- Update `README.md` with run instructions and demo command.
- Coordinate merges / assign reviewers.

Deliverables / Acceptance criteria

- `assemble()` function that reads frames + audio and outputs playable MP4.
- One end-to-end example run that produces `demo_output.mp4` using placeholder visuals and generated audio.
- A passing integration test that asserts the output mp4 file exists and basic duration > 0.
- CLI command documented in README.

Branch name

- `feature/assembler-orchestrator`

Batool — Parser & Data Schema (owner: `parser`, scene format, sample scripts)

Files / folders to own

- `src/pipeline/parser.py` — implement parser that converts raw script text → `scenes.json`.
- `data/sample_scripts/` — add more examples and edge-case scripts.
- `docs/scene-schema.md` — document the JSON schema for scene objects.
- Unit tests in `tests/test_parser.py`.

Deliverables / Acceptance criteria

- Parser CLI: `python -m ttv.parser --in data/sample_scripts/demo.txt --out data/sample_scripts/demo.json`
- Output schema includes: scene_id, start_cue (optional), description, characters, dialogue, visual_prompts (explicit), duration_hint.
- Unit tests: at least 3 tests covering normal, multi-scene, and malformed input (parser should fail gracefully with clear error).
- Provide example `scenes.json` in `data/` for the demo script.

Branch name

- `feature/parser`

Member C — Visual Generation Engineer (owner: `visual_gen`, frame renderer, caching)

Files / folders to own

- `src/pipeline/visual_gen.py` — wrapper with function `generate_frames(scenes_json, out_dir)` that outputs numbered PNG/JPEG frames.
- `assets/visual_templates/` — simple templates (PIL-based) as fallback so end-to-end runs even without heavy models.
- Unit tests in `tests/test_visual_gen.py`.
- Add caching logic: if prompt already rendered, reuse file.

Deliverables / Acceptance criteria

- `generate_frames()` must accept `scenes.json` and produce one folder per scene with frame files (e.g., `scene_01/frame_0001.png`).
- Fallback implementation that uses PIL to render text + background when model not available (keeps project zero-cost/self-hosted).
- One sample set of frames for the demo script committed under `assets/` or generated during CI.
- Tests assert correct number of frames and valid images.

Branch name

- `feature/visual-gen`

Member D — Audio / TTS & DevOps (owner: `audio_synth`, Docker/CI, docs)

Files / folders to own

- `src/pipeline/audio_synth.py` — TTS wrapper with function `synthesize(text, out_path, voice=...)`.
- `docker/Dockerfile`, `infra/docker-compose.yml` — containerize sample run.
- Update `.github/workflows/ci.yml` to run lightweight checks and end-to-end sample (using fallback visuals/TTS).
- `docs/setup.md` — how to set up environment (Windows PowerShell venv, Linux instructions).
- Unit tests in `tests/test_audio.py`.

Deliverables / Acceptance criteria

- `audio_synth.synthesize()` must produce a valid WAV (or MP3) from text input using a free/offline TTS library (if the real model is not yet integrated, create a small fallback using `pyttsx3` or `gTTS` fallback instructions — ensure no paid services).
- Provide Dockerfile that runs a minimal pipeline (parser → visual fallback → audio fallback → assembler) producing `demo_output.mp4`.
- CI workflow updated to run `python -m ttv --demo` or similar and validate output artifact existence.
- Tests validate audio file exists and basic duration > 0.

Branch name

- `feature/audio-devops`