Migration Report: CSV/JSON Storage to SQLite

Project: App-sensibilisation-video

November 7, 2025

Contents

1 Purpose

This living document records each step of migrating the project from ad-hoc CSV (`data.csv`) and JSON (`data.json`) storage to a structured SQLite database using the `better-sqlite3` Node.js library. It will be updated after every migration task (schema changes, data import, service refactors, etc.).

2 Baseline State (Pre-Migration)

- User session data appended to `data.csv` with 17 columns (user choices, resolutions, metadata).
- Aggregated score/time data persisted in `data.json` (nested objects for scores, times, precisions).
- No database; all queries implied linear scans / in-memory aggregation.

3 Step 1: Environment Setup (Completed)

3.1 Objective

Add a performant, synchronous SQLite driver to project dependencies.

3.2 Actions

1.1 Installed dependency: `better-sqlite3@Î1.10.0`.
Added to `package.json` under `dependencies`.

1.2 Updated `package-lock.json` (36 packages added transitively).[1]

1.3 No other scripts modified at this step.

3.3 Result

Project is now capable of creating and interacting with a local SQLite database file. No runtime code yet consumes the DB.

4 Step 2: Database Initialization (Completed)

4.1 Objective

Introduce a reproducible schema creation script and persistent database file.

4.2 Artifacts Created

- Directory: `db/`
- Script: `db/init-database.js`
- Database file (after execution): `db/database.db`
- Added npm script: `d̈b:iniẗ:  n̈ode db/init-database.js¨`

4.3 Schema Defined

Two tables established:

[1]As reported by npm: "added 36 packages, audited 115 packages, 0 vulnerabilities".

sessions Mirrors original CSV structure; adds surrogate primary key for internal references.

```
CREATE TABLE IF NOT EXISTS sessions (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  user TEXT NOT NULL,
  category1 TEXT,
  videoName1 TEXT,
  videoPath1 TEXT,
  resolution1 TEXT,
  category2 TEXT,
  videoName2 TEXT,
  videoPath2 TEXT,
  resolution2 TEXT,
  QO1 TEXT,
  QO2 TEXT,
  QO3 TEXT,
  QO4 TEXT,
  QO5 TEXT,
  comments TEXT,
  screenType TEXT,
  timestamp TEXT NOT NULL
);
CREATE INDEX IF NOT EXISTS idx_sessions_user ON sessions(user);
CREATE INDEX IF NOT EXISTS idx_sessions_timestamp ON sessions(
    timestamp);
```

users Stores aggregate metrics previously in JSON.

```
CREATE TABLE IF NOT EXISTS users (
  pseudo TEXT PRIMARY KEY,
  totalScore INTEGER DEFAULT 0,
  totalTime INTEGER DEFAULT 0,
  sessionCount INTEGER DEFAULT 0
);
```

4.4 Pragmas

Enabled Write-Ahead Logging and foreign key enforcement:

```
PRAGMA journal_mode = WAL;
PRAGMA foreign_keys = ON;
```

4.5 Verification

Execution output confirmed creation of tables: `sessions`, `users`, plus `sqlite_sequence` (auto-increment bookkeeping).

5 Step 3: Data Migration (Completed)

5.1 Objective

Preserve historical data by importing existing flat files into the new SQLite schema.

5.2 Artifacts Created

- Script: `db/migrate-data.js` (one-time migrator).
- Backups: `backups/data.csv.bak`, `backups/data.json.bak`.
- NPM command: `migrate:data` to run the migrator.

5.3 What the script does

- Reads `data.csv` and maps each row to the `sessions` table.
- Reads `data.json` and aggregates per-user totals into `users`.
- Wraps inserts in a single transaction; enables WAL and sets `synchronous=NORMAL` for faster batch writes.
- Supports a dry run via `DRY_RUN=1` to validate counts without writing.
- Uses an UPSERT for users so the script can be re-run safely without duplicating rows.

5.4 Field mapping

CSV (sessions) The following columns are mapped 1:1 from the CSV header to the table columns:

```
user, category1, videoName1, videoPath1, resolution1,
category2, videoName2, videoPath2, resolution2,
QO1, QO2, QO3, QO4, QO5,
comments, screenType, timestamp
```

If a field is missing in a row, it is inserted as NULL (except `user` and `timestamp`, which default to 'unknown' and current ISO time respectively).

JSON (users) The source file combines three objects: `scores`, `times`, and `precisions`. The script builds one record per pseudo as:

```
pseudo, totalScore <- scores[pseudo]
        totalTime  <- times[pseudo]
        sessionCount <- precisions[pseudo].sessions
```

Missing values default to 0. The users table is updated with an UPSERT so totals reflect the JSON content exactly.

5.5 How we ran it

Backups were created first and then the migration was executed:

```
mkdir -p backups
cp data.csv backups/data.csv.bak
cp data.json backups/data.json.bak

# Preview (no writes)
DRY_RUN=1 npm run migrate:data

# Execute migration
npm run migrate:data
```

5.6 Verification

After the run, we verified row counts using Node with `better-sqlite3`:

```
node -e "const Database=require('better-sqlite3');\
const db=new Database('db/database.db');\
console.log('sessions:',db.prepare('SELECT COUNT(*) AS c FROM
    sessions').get().c);\
console.log('users:',db.prepare('SELECT COUNT(*) AS c FROM users').
    get().c);\
db.close();"
```

Observed on this run: **116** sessions and **29** users.

5.7 Idempotency and safety

- **Users**: UPSERT ensures re-running sets totals to JSON values rather than duplicating.
- **Sessions**: Designed for one-time import; re-running would re-insert sessions. Avoid re-running or add a guard file if needed.
- **Dry run**: Use `DRY_RUN=1` to validate counts before writing.

6 Change Log

Step 1
Added dependency `better-sqlite3`. (No code paths yet updated.)

Step 2
Created schema initialization script; introduced `sessions` and `users` tables; added npm script `db:init`.

Step 3
Migrated historical data with `db/migrate-data.js`; added npm script `migrate:data`. Verified counts match sources.

7 How to Reproduce Current DB State

Run:

```
npm install
npm run db:init
```

This will (re)create the database with the current schema.

Populate from existing CSV/JSON (optional) Run the one-time migration to import current `data.csv` and `data.json`:

```
# Dry run (no writes):
DRY_RUN=1 npm run migrate:data

# Execute migration:
npm run migrate:data
```

Afterwards, the database contains the historical sessions and users aggregates.

This document will be updated after each subsequent migration task.