

Worked Software Project Example: A Small CSV Inventory Summary Tool

A teaching report grounded in a runnable Node.js starter, its synthetic fixture and passing tests.

What this document is. A worked explanation of the exact educational software package distributed with it: a command-line program that reads a two-column CSV inventory, validates each row and reports summary values.

What it is not. This is not a student submission, completed final-year research project, production stock-control system, or empirical study of users or businesses. The example inventory is fictional. This report records code behavior and automated test evidence only.

At a glance

Item	Documented example
Problem boundary	Demonstrate a small file-processing workflow with explicit input rules and useful failure messages.
Implementation	Node.js ES modules, built-in file reading, local CSV parser, summary function and Node test runner.
Evidence	Three automated tests pass; supplied CLI fixture prints four rows, total quantity 13 and two low-stock items.
Data status	Four fictional stationery rows; no people, institution, customer records or observations.
Use	Read with the ZIP starter. Adapt only after defining an approved requirement and your own acceptance tests.

How to use this report

Pages 2-4 define the example brief, technical background and input contract; pages 5-7 trace implementation and reproduced evidence; page 8 states limitations; pages 9-10 provide adaptation prompts and source notes. The source archive is authoritative. If an extracted copy differs, rerun its tests and update this report before relying on it.

Version context: inspected from the supplied ZIP on 30 September 2026; local validation used Node.js 24.11.1. The package declares Node.js 20 or later and has no external dependencies.

PROJECT BRIEF

1. Project introduction and bounded problem

Small organisations sometimes track a short inventory in a spreadsheet or CSV file. A learner may want to understand how a program can read such a file, reject malformed values and produce a quick summary. This teaching starter makes that narrow programming workflow concrete. It does not claim that a particular organisation has an inventory problem or that this script improves a real operation.

Problem statement

A beginning programmer needs a small, inspectable example connecting file input, data validation, summary logic, command-line output and automated tests. Without an explicit input contract, a program may silently accept a wrong header, blank item name or negative stock value. The starter makes the contract visible and tests selected valid and invalid cases.

Aim and objectives

- 1. Define a CSV input contract with exact columns `item`, `quantity` and non-negative whole-number quantities.
- 2. Implement row parsing and a summary of row count, total quantity and names at or below the fixed threshold of two.
- 3. Provide a CLI that reads a caller-supplied file path and prints a compact summary or a clear error.
- 4. Use automated tests to check the example inventory, quoted values and selected invalid inputs.
- 5. Document the example data as fictional and identify extension points and limitations.

Scope

Included	Outside the supplied implementation
One local CSV file per invocation.	Database, API, web UI, accounts or persistent storage.
Two named fields, quoted commas and doubled quotes on one physical line.	Multi-line quoted CSV fields or complete RFC 4180 enforcement.
Summary values and fixed ≤ 2 low-stock rule.	Configurable thresholds, duplicate detection, replenishment.
Built-in test runner and three test cases.	User study, deployment, security audit or production readiness.

The boundary is intentional: this is a testable program slice rather than a complete business system. A real assignment may require different deliverables; check with the supervisor.

OFFICIAL TECHNICAL REFERENCES

2. Technical background and design rationale

Node.js runtime and modules

The package uses JavaScript ES modules through `"type": "module"`. File operations use `node:fs/promises`; the CLI awaits a UTF-8 file read and calls the summary function. The built-in test runner executes tests with `node --test` without another testing dependency. These are runtime capabilities, not proof that every supported runtime behaves identically on every operating system [1-3].

CSV is a convention with edge cases

RFC 4180 documents a common CSV form: comma-separated fields, records separated by line breaks, and double-quoted fields when values include commas, line breaks or quotes. It is informational and notes implementation variation [4]. This starter handles a subset: one physical line per record, double-quoted values, commas inside those quoted values and doubled quote escapes. It does not parse a quoted field spanning multiple lines and does not enforce every quote-placement rule in the RFC.

Why a small custom parser

The goal is to expose a parsing state machine rather than hide it behind a dependency. The parser tracks whether it is inside a quoted field. A comma ends a field only outside quotes; two adjacent quotes inside a quoted field become one literal quote. An open quote at line end is treated as malformed input. This fits the included examples, with the scope limits stated above.

Verification and validation

NASA software guidance distinguishes checking whether requirements describe the right product from checking whether implementation meets requirements; its handbook also emphasizes planned tests and recorded evidence [5]. The starter borrows this distinction as a teaching idea, not as a claim of compliance with NASA processes. Here, unit tests verify concrete program behavior. They do not validate that an inventory tool meets the needs of a real storekeeper.

The appropriate engineering conclusion is modest: the source makes input and output rules inspectable, and the tests cover selected cases. Suitability for a real setting remains unassessed.

INPUTS, OUTPUTS AND ACCEPTANCE EVIDENCE

3. Requirements and data model

ID	Requirement	Evidence in supplied package
R1	Require exact header item, quantity.	Header parsed and checked; invalid-header test.
R2	Require two fields per data row and non-empty item name.	Field-count and name checks in summarizeInventory.
R3	Require non-negative integer quantity.	Number conversion and range check; negative-quantity test.
R4	Return row count, quantity sum and names <= 2.	Pure function; fixture test expects 4, 13 and two names.
R5	Allow commas and doubled quotes in supported quoted fields.	Dedicated quoted-field test.
R6	Provide a CLI accepting a file path.	node src/index.mjs examples/inventory.csv.

Data dictionary: fictional fixture

Field	Type / rule	Example	Interpretation
item	Text; trimmed; non-empty.	Blue pen	Fictional stationery name.
quantity	Non-negative integer.	2	Invented count for exercise only.

Synthetic example rows

item	quantity	Result under supplied rule
Blue pen	2	Low stock because 2 <= 2.
Exercise book	6	Not included.
Graph paper	1	Low stock because 1 <= 2.
Pencil	4	Not included.

This fixture is illustrative content, not an observed business dataset. Its result describes four supplied rows only. It cannot support an estimate of real demand, an operational decision, a customer claim or a finding about inventory practices.

ARCHITECTURE

4. Implementation structure and data flow

```
node-project-starter/  
  package.json  
  README.md  
  LICENSE  
  examples/inventory.csv  
  src/index.mjs  
  test/inventory.test.mjs
```

Execution path

1. The CLI reads the path from `process.argv[2]`; if absent, it prints usage and sets a non-zero exit code.
2. `readFile` loads UTF-8 text from the resolved local path. File errors are caught and printed.
3. `summarizeInventory` trims input, splits physical lines, parses the header and validates each row.
4. The function returns `rowCount`, `totalQuantity` and `lowStockItems`.
5. The CLI prints those values. It does not modify or write back to source data.

Core behavior excerpt

```
return {  
  rowCount: items.length,  
  totalQuantity: items.reduce((total, item) => total + item.quantity, 0),  
  lowStockItems: items.filter((item) => item.quantity <= 2).map((item) => item.name),  
};
```

The pure function takes a string and returns a summary or throws a validation error. Keeping calculation separate from disk access lets tests exercise parsing without filesystem setup, while one test reads the included fixture to verify the input-to-summary contract.

Dependencies and license

The starter has no third-party runtime or test dependencies. Package metadata marks it private and includes an MIT license for the supplied source. That does not grant rights to later-added data or code; check each dependency and institution policy when extending the scaffold.

OBSERVED OUTPUT FROM SHIPPED FIXTURE

5. Reproduced command-line example

The archive was extracted to a temporary directory and run with Node.js 24.11.1. This command reads only its bundled fictional fixture:

```
node src/index.mjs examples/inventory.csv
```

Observed stdout:

```
Rows: 4  
Total quantity: 13  
Items at or below quantity 2: 2  
Low stock: Blue pen, Graph paper
```

How to interpret it

Rows: 4 counts data records after the header. **Total quantity: 13** adds $2 + 6 + 1 + 4$. **Items at or below quantity 2: 2** applies the code's fixed comparison and returns Blue pen and Graph paper.

These figures are calculations from the package fixture, not research results. The threshold is a demonstration constant. A user would need to define a meaningful rule for a real process and test the change; the current CLI has no option to configure it.

Repeat the run

1. Extract the ZIP and open a terminal in `node-project-starter`.
2. Confirm Node.js is installed with `node --version`; package minimum is 20.
3. Run `node src/index.mjs examples/inventory.csv` and compare output above.
4. Run `npm test` to use Node's built-in test runner.

Matching output demonstrates repeatability for this fixture in the tested environment. It does not constitute a field evaluation or prove behavior on every CSV dialect or system.

VALIDATION RECORD

6. Automated tests and evidence boundary

The test file contains three cases. Running `npm test` in the extracted package under Node.js 24.11.1 produced three passes and zero failures.

Test case	What it checks	Observed expectation
Synthetic fixture summary	Reads package CSV and checks exact summary object.	4 records; sum 13; two low-stock names in order.
Quoted fields	Comma inside quoted item and doubled quotes inside quoted item.	Two records; total 5; literal quote retained.
Invalid inputs	Negative quantity, wrong header, unterminated quoted value.	Each throws an error matching the expected validation message.

Command and observed summary

```
npm test

tests 3
pass 3
fail 0
```

What passing tests establish

The assertions pass against this package version and current Node runtime. They protect included summary behavior and selected parser boundaries. They do not establish requirement completeness, usability, security, reliability under large files, compatibility with all CSV readers or fitness for a real organisation.

Test gaps to address before extending

- Blank file, header-only file, blank rows, whitespace quantities and extra columns.
- Quotes in invalid positions, multiline quoted fields and strict CSV grammar.
- Very large files, duplicate names, non-ASCII text and platform-specific path behavior.
- End-to-end CLI exit codes and stderr for missing arguments and unreadable paths.

An extension should add a test for each agreed behavior before reporting it as implemented. Keep new tests, observed outputs and scope changes in the project record.

WHAT REMAINS UNTESTED

7. Limitations, assumptions and safe use

Area	Current limitation	Possible next check
CSV grammar	Splits on physical newlines; not full RFC 4180. Quote placement is not strictly validated.	Adopt a maintained parser or define and test a dialect.
Input scale	Reads entire file into memory and processes all rows.	Set a size limit; benchmark representative synthetic fixtures.
Business meaning	Threshold ≤ 2 is fixed and illustrative; duplicate names count separately.	Ask an authorized user to define units, threshold, duplicate policy and output.
Persistence	No database, edit, audit log, backup or authorization.	Specify access control, retention and source of truth before adding storage.
Evaluation	No people or operational setting were involved.	If human evaluation is appropriate, confirm ethics and site permission first.

Assumptions

- Input has a header, at least one record and exactly two columns.
- Each physical line is a record; line breaks do not occur inside fields.
- Quantities are whole numbers and totals remain within JavaScript number precision for this small fixture.
- Fixture item names are distinct, although the code does not require uniqueness.
- Example records are synthetic and contain no personal or confidential data.

Ethical and practical boundary

Do not replace the fixture with a real organisation's stock file without permission. Operational files can disclose commercially sensitive information or personal details. Minimise copied fields, use approved storage, and follow relevant data-protection rules. A class project or dissertation may also require supervisor and ethics review before collecting feedback from people.

The included MIT license concerns the supplied source. It does not approve academic reuse or replace attribution and disclosure required by a course. Report only code you wrote or changed, distinguishing inherited scaffold behavior from your own work.

USE THE EXAMPLE AS A SCAFFOLD FOR REASONING

8. Conclusion and adaptation prompts

This worked report demonstrates a limited software lifecycle slice: a requirement, two-field input contract, parser, validation, summary logic, command-line interface, fixture, documentation and three automated tests. The archive is runnable and its output can be reproduced. That is concrete teaching material, not evidence of a completed research study or a production-ready tool.

A disciplined extension sequence

1. Write a new problem statement for a specific context you are authorized to study.
2. Ask a supervisor or intended stakeholder to review the question and acceptance criteria.
3. Decide how duplicate names, blank records and threshold values should work.
4. Choose a CSV dialect and parser strategy; document unsupported inputs.
5. Add focused tests for every changed requirement and rerun the suite.
6. If a real setting or people are involved, secure permissions and ethics review before collecting data.
7. Report what was changed, tested and left out, with citations and a versioned project record.

Adaptation prompts

- What evidence shows that a proposed user needs a summary, and what alternative workflow should be compared?
- How would you parameterize the threshold without making the CLI confusing?
- What should happen when item names repeat or quantity is 2.5?
- Which CSV edge cases matter for the intended users and data source?
- What evidence could test correctness without exposing confidential inventory data?
- What constraints or maintenance needs would change for a web application?

A strong report answers these questions with verified context and observed evidence. The prompts are not results and should not be copied as if research had been conducted.

PRIMARY TECHNICAL REFERENCES

References and source notes

- [1] Node.js Documentation. "ECMAScript modules."
<https://nodejs.org/api/esm.html>
- [2] Node.js Documentation. "File system - node:fs."
<https://nodejs.org/api/fs.html>
- [3] Node.js Documentation. "Test runner."
<https://nodejs.org/api/test.html>
- [4] Shafranovich, Y. RFC 4180: Common Format and MIME Type for Comma-Separated Values (CSV) Files. RFC Editor, October 2005. Informational memo; it describes a common form and notes implementation variation.
<https://www.rfc-editor.org/info/rfc4180/>
- [5] NASA Software Engineering Handbook. "SWE-055 - Requirements Validation." Cited for the verification/validation distinction, not as a student-project standard.
<https://swehb.nasa.gov/spaces/7150/pages/16449673/SWE-055%2B-%2BRequirements%2BValidation>

Source and reproducibility notes

Technical background was checked against the primary references on 30 September 2026. The CSV fixture, source and tests were read from the downloadable ZIP. The test and CLI outputs were rerun from an extracted copy before writing this report. No participant responses, real inventory records, financial projections or fabricated study findings appear here.

Suggested citation: FINAL YEAR NG. (2026). *Worked Software Project Example: A Small CSV Inventory Summary Tool* [Educational worked report and runnable source archive]. Cite and disclose adaptation according to your programme requirements.

The reference list supports technical explanations only. Local project format, software requirements, ethics approval and submission rules remain the responsibility of the institution and supervisor.