**Planning and requirements analysis**

Goal: check whether the detergent is being used by other people besides our family.

Metric of achieved goal: the tool can accept inputs, catch invalid config and calculate a usage summary1 correctly.

Why: the detergent runs out suspiciously quickly, so a (semi) automated solution will be used to confirm / deny the suspicion.

What: mini CLI tool that tracks how much detergent has been used up and calculates a usage summary.

Users: developers who face the same problem

Features:

* (Persistent) config file with:
  + Bottle volume
  + Cup volume
* (Persistent) database
* Logging: how much detergent has been used in the current wash (in cups)
* Status report: usage summary

The usage is logged after launching a wash, so an entry per wash cycle

No resources other than manual input will be used.

The app won’t forecast when to buy more detergent or when the current one will end. Neither will the app calculate descriptive or any other kind of statistics (mean, time intervals, etc.).

Constraints: no technical constraints beyond standard CLI environment

Assumptions: users answer honestly and log every wash without omission.

**SRS**

The purpose of the SRS is to define and explain each process, feature, and piece of logic the app shall implement.

**General description**

What: mini CLI tool that tracks how much detergent has been used up and calculates a usage summary.

Assumption: users answer honestly and log every wash without omission.

Features:

* (Persistent) config file with:
  + Bottle size
  + Cup size
* (Persistent) database
* Logging: how much detergent has been used in the current wash (in cups)
* No rollback(!)
* Status report: usage summary
* Reset the tracking
* Help

The usage is logged after launching a wash, so an entry per wash cycle

The caps for bottle and cup sizes are specified within reasonable bounds. They are subject to change if required in the future.

No resources other than manual input will be used.

The app won’t forecast when to buy more detergent or when the current one will end. Neither will the app calculate descriptive or any other kind of statistics (mean, time intervals, etc.).

**Functional requirements**

The app shall run “one way”. That is, when called it completes the task at hand (specified at call) and stops. The only persistent media are database and config.

The default error suit: Explain during what an error happened, output error name and message. A different error suit can be specified to override the default one.

The database shall be a lightweight relational database, which does not require a separate process to run.

At each launch the system shall:

* Try to read the config file. On error use the default suit + suggest looking at the github page to get the default config file without resetting the whole system.
* Check that only the required keys are present. On error default suit.
* Check that bottle size is a real number and bottle size > 0. On error use the default suit.
* Check cup size > 0. On error use the default suit.
* Load the database. On error use the default suit.

Check if data table exists. If not create a table with

* Date (dtype date)
* N\_cups (dtype float)
* Amount\_detergent (dtype float)

Log the creation in the terminal.

The allowed flags (and combinations) and their behaviors are:

-l (--log) <n-cups>:

* the system shall ingest the number of cups (in liters, of a certain single detergent “.” separated) used through its CLI,
* check that it’s >=0. On error use the default suit.
* the system shall calculate the volume of detergent used
* Check that volume <= bottle size. On error check the default suit
* connect to the database
* query how much detergent should be left
* check the volume used from user input >= volume left. On error use the default suit
* insert date, number of cups, and amount of detergent used into a database alongside with the current date.
* Return “Logged successfully.”

-s (--status):

* Connect to the database
* query the database for:
  + first entry date
  + sum of detergent used (liters)
* Get the total size of the bottle
* calculate the (expected) percentage used (round to 2 decimals)
* Return an output, which shall include:
  + The first entry date
  + The expected percentage used
  + Liters used / liters total indicator (in brackets)
  + A bar spanning the entirety of the terminal window which shall indicate an (expected) amount of detergent left. Whereby a solid character means presence and non-solid (whitespace) – the lack of detergent.
  + Status bar algorithm as follows:

The output is width of the terminal with 2 closing characters and terminal width – 2 indicator characters long. The closing character shall differ from indicator / empty character and resemble closeness (like brackets). The indicator character shall be solid and empty characters whitespaces.

The system shall take in the percentage load (0 <= x <= 1), calculate the number of indicator characters and the number of whitespace characters and output them like [##... whitespaces].

-l (--log) <n-cups> -s (--status):

* Joined log and status check
* The system shall do first the steps from log and then from status

--reset (explicitly no shortened version):

* Prompt user whether they want to delete data, on yes proceed, on no quit
* the system shall delete the database
* reset the config file to default
* return a success message. Error out on failure

--help:

* The system shall return the purpose of the app and the user’s manual
* The manual shall include the following:
  + Mention that only certain flags are allowed
  + List them with explanations
  + Mention the need to fill the config file

Any other flag / combination shall result in (explicit) error: “An illegal flag / combination has been provided: <user-input>. Please user (--help) to see the user’s manual”.

If any unaccounted-for error occurs, output: “An unexpected error occurred: <error> – <error-message>”.

**Interface requirements**

All the communication takes place over the terminal, in English language.

Every volume measurement is in liters.

There shall be explicitly no external I/O (besides the one done by libraries under the hood).

The system shall be a monolith, so no additional commutation (besides the one happening in libraries) shall take place.

**Performance requirements**

The app shall not take more than a second to execute a specified task (log or status or anything else specified in functional requirements). This will probably always hold since the input and operation are trivial.

**Design constraints**

The app shall be a (containerized) monolith, running on a single machine. The database shall be configured considering this.

No technical constraints beyond standard CLI environment.

**Non-functional requirements**

The app shall be cross-platform

**Preliminary schedule**

Best-case scenario: 28.09.2025

Expected case scenario: 29.09.2025

Worst case scenario: 03.10.2025

**DDS**

**HLD**

Tech stack:

* Base language – python
* Argument parsing – built-in argparse library
* Database – SQLite (built-in sqlite3 library)
* Config handling – JSON (built-in json library)
* Containerization – Docker

Module structure:

* /detergent\_tracker
  + CONFIG.json
  + default\_config.json
  + database/ (to make mounting a named volume later easier)
    - detergent\_records.db
  + main.py
  + README.md
  + SDLC.docx

(In development phase the root is the project folder)

Later Docker related files will be added.

Git files are not mentioned and should be ignored.

Gitignore should include everything besides

* default\_config.json
* main.py
* README.md
* SDLC.docx

**LLD**

On any manually raised error quit the app.

On unaccounted-for errors raise RuntimeException “An unaccounted-for error occurred: <error> – <error-text>. Traceback: <traceback>. Contact the maintainer for help at [molego16@gmail.com](mailto:molego16@gmail.com).” Use sys(.excepthook) and traceback modules to implement.

Config must have (only) the following keys:

* Bottle\_volume: float
* Cup\_volume: float

At launch:

* Try to json.load(<path-to-config>). On error raise ImportError “An error occurred while reading config: <error> – <error-text>. Visit <github-link-to-repo> to see the manual on how to handle the config and its default contents”.
* Check the number of keys against the expected and the keys themselves against the expected values. On error raise ValueError “An error occurred while parsing config file: there are more keys (<n-keys>) than expected (<n-expected>)” or ValueError “An error occurred while parsing config file: the expected keys (<expected-keys-coma-separated>) don’t match the actual ones: <actual-keys-coma-separated>”
* Iterate over keys and check the values
* bottle size is a real number and bottle size > 0. On error TypeError “The bottle size must be a real number, got <got-size>” if the size is not numeric. ValueError “The size must be bigger than 0, got <got>”, if the size is out of limits.
* Same check and error messages for the cup size (cup size > 0)
* Load the config into heap with variables of the same names
* Check if the database directory exists. If the directory does not exist raise FileNotFoundError “The database directory does not exist. Make sure you create a ‘database/’ directory or mount a named volume ‘database’ when launching the container”.
* Try to load the database. If the DB does not exist print “Database not found creating a new one”.
* Create a new database in database/ directory. Print database created. On error RuntimeError “An unexpected error occurred while trying to create the database: <error-name> – <error-text>”.
* Load the database. On error ImportError “Failed to load the database: <error> – <error-text>”

Check if data table exists. If not create a table with

CREATE TABLE IF NOT EXISTS usage\_data (

date INTEGER NOT NULL DEFAULT strftime('%s', 'now'),

n\_cups REAL NOT NULL,

volume\_used REAL NOT NULL,

PRIMARY KEY (date ASC) ON CONFLICT FAIL,

CONSTRAINT valid\_cups CHECK n\_cups >= 0,

CONSTRAINT valid\_valid\_used CHECK volume\_used >= 0

)

Log the creation in the terminal.

Any flag / combination not specified below will raise a ValueError: “An illegal flag / combination has been provided: <user-input>. Please user (--help) to see the user’s manual”.

-l (--log) <n-cups>:

* Parse the flag
* Try to cast to float. On error raise ValueError “An error occurred while parsing the number of cups, expected float, got <got>. Use full stop to separate decimals and don’t use any separator for integer parts”
* check that it’s >=0. On error raise ValueError “Invalid number of cups. Got <n-cups>, expected a float >= 0”
* calculate the volume used: n-cups \* cup-volume, save it as a variable
* Check that volume currently used <= bottle size. On error raise ValueError “The volume most recently used (<n-cups> \* <cup-volume> = <volume>) exceeds the total volume of the detergent <bottle-volume>. You may need to adjust cup and / or bottle volumes and relaunch the app.”.
* connect to the database with sqlite3.connect(<absolute-path>). On error re-raise the error with “Connection to the database failed: <error> - <error-message>”.
* query how much detergent should be left:

<bottle-volume> – (SELECT SUM(volume\_used) FROM usage\_data) >= <most-recent-volume>

On error raise ValueError “The most recent volume of detergent exceeds the (expected) volume left”

* Insert data with

INSERT INTO usage\_data (n\_cups, volume used) VALUES (<n-cups>,<volume-used>)

On Error SystemError “Unable to insert data: <error> - <error-text>. Try again later”.

Return “Recorded successfully.”

-s (--status):

* Check if the database file exists. If not output:

As of <current-date>, 0% (0/ <bottle\_volume>) of the detergent should have been used.

The bar (from playground.ipynb) at 0%

* connect to the database with sqlite3.connect(<absolute-path>). On error re-raise the error with “Connection to the database failed: <error> - <error-message>”.
* query the database:

SELECT

SUM(volume\_used) AS total\_used,

MIN(date) AS first\_entry

FROM usage\_data;

* calculate total\_used / bottle\_volume as pct\_used
* Output the following into the terminal

First use of the current detergent on <first-entry> (converted from timestamp to DD.MM.YYYY).

As of <current-date>, <pct\_used>\*100% (<total\_used> / <bottle\_volume>) of the detergent should have been used.

The bar (from playground.ipynb) at <pct\_used>.

-l (--log) <n-cups> -s (--status):

* Call the -l flag
* Commit the DB
* Call the status flag

--reset (explicitly no shortened version):

* Prompt user input(“Do you really want to reset all the data [n/Y]: ”)
* If n: quit
* Delete the database file
* Write the default\_config.json to the CONFIG.json, if does not exist, create and write
* Log “Successfully deleted the data.”. On error RuntimeError “An unexpected error occurred while deleting the data: <error> – <error-message>”.

--help:

The following text should be returned:

“This app has been created to track usage of detergent.

Before using the app, I strongly advice to fill the config file and attach it, otherwise the app will work in an unexpected way or not run at all. The following options are available:

* Bottle\_volume: float > 0 (decimal “.” Separated, not integer separator)
* Cup\_volume: float > 0 (decimal “.” Separated, not integer separator)

The following features are implemented:

* Enter the number of cups of detergent used in the last wash. Call the app with -l (--log) <n-cups> flag to input the data
* See the current (expected) usage of the detergent with -s (--status) along with the date of the first usage
* Delete all the data with --reset.
* To get help (you are getting it right now) call with --help flag

Advise README.md to learn the run procedure.”

README.md content

“This app has been created to track usage of detergent.

Before using the app, I strongly advice to fill the config file (CONFIG.json) and attach it, otherwise the app will work in an unexpected way or not run at all. The following options are available:

* Bottle\_volume: float > 0 (decimal “.” Separated, not integer separator)
* Cup\_volume: float > 0 (decimal “.” Separated, not integer separator)

Adding other options will result in an error.

Run the app with <run-command> + your flags & values.

For example: <example command> to get status

The following features are implemented:

* Enter the number of cups of detergent used in the last wash. Call the app with -l (--log) <n-cups> flag to input the data
* See the current (expected) usage of the detergent with -s (--status) along with the date of the first usage
* Delete all the data with --reset.
* To get help (you are getting it right now) call with --help flag

The data format is DD.MM.YYYY”.

**Containerization**

The production shall be run as a single container.

Create /detergent\_tracker inside the container as home folder.

Handle the files in the following manner:

* README.md – ignore (no need for readme in the container)
* CONFIG.json – bind mount volume (for fast adjustments without rebuilds)
* default\_config.json – copy
* (/detergent\_tracker/)database/ - create
* detergent\_records.db – ignore, mount a named volume (/app/database/) to create a DB
* main.py – copy

Use python-alpine as base image.

**Development**

Hello world!

**Appendices**

1 – usage summary always refers to:

* when the current detergent has been started (date)
* how much detergent has been used (total)
  + as a nice colored bar (where the bar goes down as more detergent is used) and
  + a numeric value (in L) and in % of the whole bottle.