# Improving the **spectra‑app** using Astropy & Astroquery

## 1. Introduction and current issues

Your **spectra‑app** displays astronomical spectra by ingesting FITS files, reading metadata and overlaying them in Streamlit. The current implementation suffers from several problems:

* **Ad‑hoc import logic:** the ingestion code attempts to parse many file types with custom logic and falls back to astropy.io.fits manually. This leads to unhandled warnings about WCS axes and duplicate CDELT keywords and generates RuntimeWarning messages in the console.
* **Unstable asynchronous downloads:** the app defines the OverlayIngestResult class inside the Streamlit script. When Streamlit reruns the script, the class identity changes and asynchronous futures completed by a previous run are no longer recognised. This triggers the Unexpected ingest result state described by Codex and prevents ingestion results from being forwarded to the overlay manager.
* **Unreliable data sources:** your script scrapes many stars from random sources. Many FITS files are empty or incomplete; some contain 3‑D data or missing HDU definitions, leading to FITSFixedWarning messages and ingestion failures.
* **Inconsistent units and coordinate handling:** wavelength arrays are often stored in different units (Ångström, nanometer, m/s) and the app guesses units rather than reading them explicitly. RA/Dec positions may be stored in text or not at all, making cross‑matching difficult.
* **Confusing library tab:** the library tab presents a long, uncurated list of example stars fetched by a script. Users must scroll through numerous broken entries and cannot upload their own spectra easily.

Below are actionable recommendations based on the Astropy/Astroquery documentation and your repository, along with proposals for code refactoring. Citations are provided from the documentation for key features and constraints.

## 2. Improve data import and format handling

### 2.1 Use Astropy’s unified I/O interface

Astropy’s high‑level read()/write() methods provide a consistent way to read tables, images and NDData objects across many formats. The **unified file I/O** interface automatically determines the file type from the filename extension and can handle FITS, VOTable, ECSV, HDF5, Parquet, PyArrow CSV, etc. The docs note that the high‑level interface is easy to use and recommended over low‑level FITS routines[[1]](https://docs.astropy.org/en/latest/io/overview.html#:~:text=Astropy%20provides%20two%20main%20interfaces,for%20reading%20and%20writing%20data). For example, reading a FITS table into a Table is as simple as:

from astropy.table import Table  
# explicit format avoids guessing overhead  
tbl = Table.read('my\_spectrum.fits', format='fits', hdu='SCI')

When writing, the format can be inferred or specified. The help() string on the Table.read method can be used to discover supported formats[[2]](https://docs.astropy.org/en/latest/io/overview.html#:~:text=Use%20,returns%20a%20QTable). Encouraging all ingestion functions to use Table.read or CCDData.read (for images) ensures that units, column mixins and metadata are reconstructed properly[[3]](https://docs.astropy.org/en/latest/io/unified_table_fits.html#:~:text=FITS).

**Recommendation:** wrap all file ingestion in functions that call Table.read/NDDataRef.read with an explicit format (FITS, ASCII, etc.) and hdu parameter. Catch exceptions from astropy.io.ascii for unsupported text formats and log meaningful errors. This removes the need for manual FITS header parsing and reduces the warnings due to WCS mismatches. Use the high‑level interface for writing user uploads back to disk.

### 2.2 Represent data with NDDataRef to unify arrays and metadata

The astropy.nddata subpackage provides NDData and NDDataRef, which act as containers for n‑dimensional data with associated metadata, unit, uncertainty, mask and WCS. NDDataRef extends NDData with read/write methods that tie into the unified file I/O and support arithmetic and slicing[[4]](https://docs.astropy.org/en/latest/nddata/index.html#:~:text=,%E2%80%99s%20unified%20file%20I%2FO%20interface). Using NDDataRef objects for spectral arrays ensures that each dataset carries its unit, WCS and uncertainty; you no longer need to store separate dictionaries for metadata.

**Recommendation:** after reading a FITS file, wrap the data into an NDDataRef (if it is an image) or keep it as an astropy.table.QTable (if it is a 1‑D spectrum). Store these objects in your session state instead of raw numpy arrays. When plotting, extract data and unit directly. This allows consistent unit conversion and simplifies later export to other formats.

### 2.3 Handle units explicitly using astropy.units

Astropy’s units.Quantity combines numerical values with units and supports arithmetic, conversion and decompose operations[[5]](https://docs.astropy.org/en/latest/units/index.html#:~:text=Introduction). For example, q = [500, 600] \* u.nanometer gives a wavelength array with units; calling q.to(u.Angstrom) converts to Ångström. Use this to ensure wavelengths and fluxes are always associated with a unit, and convert them to a common unit before overlaying.

**Recommendation:** when reading spectral data, convert wavelength columns to Quantity using units parsed from the FITS header (e.g., BUNIT, CUNIT1) or by default assume nanometer. Provide unit conversion options in the UI. Use to() or decompose() to transform to user‑selected units before plotting.

### 2.4 Use SkyCoord for coordinates

astropy.coordinates.SkyCoord provides a unified class for representing sky positions in different frames and units. Coordinates can be created from numeric values, sexagesimal strings or Quantity objects and specify frames such as ICRS (RA/Dec), galactic or ecliptic[[6]](https://docs.astropy.org/en/latest/coordinates/index.html#:~:text=The%20best%20way%20to%20start,specified%20with%20the%20string%20name). This is especially useful when cross‑matching star positions across catalogs. You can instantiate with:

from astropy.coordinates import SkyCoord  
from astropy import units as u  
coord = SkyCoord(ra=10.625\*u.deg, dec=41.2\*u.deg, frame='icrs')

**Recommendation:** parse RA/Dec from FITS headers or user uploads into SkyCoord objects. Provide transformations to other frames as needed via coord.transform\_to('galactic'). This ensures consistent coordinate handling and makes cross‑matching with Astroquery results straightforward.

### 2.5 Suppress or handle FITS WCS warnings

Your logs show warnings like cdelt will be ignored since cd is present and 'obsfix' made the change …. These come from astropy.wcs warning about inconsistent header keywords. While warnings are not fatal, they clutter the logs. For improved robustness:

* Use astropy.wcs.WCS(header, fix=True) to let Astropy fix common header issues.
* Catch AstropyWarning or suppress specific FITSFixedWarning messages via warnings.filterwarnings('ignore', category=FITSFixedWarning) in your ingestion module after verifying that the header fixes are acceptable.
* Optionally run astropy.io.fits.verify() on incoming FITS files to catch serious issues before reading.

## 3. Selecting and fetching reliable star data

### 3.1 Query remote archives responsibly

The examples in your library fetch data from MAST, ESO, SDSS and DOI sources. The Astroquery documentation emphasises that some services enforce rate limits; SIMBAD will blacklist clients sending >5–10 queries per second and recommends submitting a vector of names via query\_objects() rather than repeated single calls[[7]](https://astroquery.readthedocs.io/en/latest/simbad/simbad.html#:~:text=A%20warning%20about%20query%20rate%C2%B6). Gaia and VizieR queries also support row‑limit parameters. Gaia.ROW\_LIMIT controls the number of rows returned (set to an integer for limited results or -1 for unlimited)[[8]](https://astroquery.readthedocs.io/en/latest/gaia/gaia.html#:~:text=Queries%20return%20a%20limited%20number,default%20behaviour%20set%20this%20appropriately), and vizier.ROW\_LIMIT defaults to 50 but can be set to -1 to retrieve full catalogs[[9]](https://astroquery.readthedocs.io/en/latest/vizier/vizier.html#:~:text=,s). Failing to set these can result in huge responses and timeouts.

**Recommendation:** build a small query layer around Astroquery that automatically:

1. Batches lists of star names with query\_objects() or resolve\_object() to reduce the number of HTTP requests.
2. Sets ROW\_LIMIT for Gaia and VizieR queries to a moderate number (e.g., 1000) and exposes it as a configuration option.
3. Includes retry logic and catches astroquery.exceptions.RemoteServiceError to handle network failures gracefully.

### 3.2 Cross‑match data and select reliable measurements

For each star, you often need data from multiple archives (spectral lines, photometry, radial velocities). Use SkyCoord to cross‑match results by position. For example, after querying Gaia and VizieR, cross‑match by RA/Dec within a tolerance using coord.separation(other\_coord) < 1\*u.arcsec. This ensures the data correspond to the same object.

### 3.3 Trust but verify: NIST Atomic Spectra Database

The NIST ASD help page explains that only data which have been *critically evaluated* by NIST are included; this provides quality reference data but means the database may lag behind the latest measurements[[10]](https://physics.nist.gov/PhysRefData/ASD/Html/help.html#:~:text=Our%20basic%20policy%20is%20to,spectra%20are%20queried%20in%20ASD). Differences between published values and NIST ASD values can occur due to updates or corrections[[11]](https://physics.nist.gov/PhysRefData/ASD/Html/help.html#:~:text=II,NIST%20published%20and%20ASD%20values). When possible, compare NIST data with other sources and note the version or date of retrieval. Provide citations or DOIs for the datasets you include.

## 4. Improving the library tab and data sources

### 4.1 Curate a small set of benchmark stars

Rather than loading every available spectrum, focus on 5–8 well‑studied stars with high‑quality, publicly available data. This allows you to design the UI around stable, reproducible datasets and avoid broken sources. A suggested initial library:

| Star | Reason for inclusion | Potential data sources |
| --- | --- | --- |
| **Sirius A** | Brightest star with HST/STIS spectra; high S/N; example already in the repo | MAST (HST/STIS), ESO |
| **Vega** | A0V standard used for calibration; multiple high‑resolution spectra available | MAST, ESO |
| **Procyon A** | Nearby F5 star with well‑studied spectrum; serves as a mid‑F standard | MAST, Gaia |
| **Betelgeuse** | Red supergiant with rich spectral features; widely studied; caution for variability | VizieR, SIMBAD |
| **HD 189733** | Bright K dwarf host star with many exoplanet studies; high‑resolution spectra | VizieR, ESO |
| **HD 80606** | Example of a metal‑rich G5 star; appears in VizieR queries[[12]](https://astroquery.readthedocs.io/en/latest/vizier/vizier.html) | VizieR, SIMBAD |
| **Polaris (α UMi)** | Cepheid variable and calibration star; accessible via SIMBAD and Gaia | SIMBAD |

These stars are widely used in spectroscopy and have numerous datasets across archives. You can add or remove stars based on your goals. For each star, create a ProviderQuery object that defines how to download the spectrum (e.g., HST/ESO product ID), default wavelength unit, and metadata.

### 4.2 Use a structured library definition

Instead of a script that scrapes arbitrary files, define the library as a JSON or YAML file listing each star, the data product IDs/DOIs, and retrieval functions. Example entry:

name: Sirius A  
provider: mast  
product\_id: "HST/STIS o8ce03cf0"  
wavelength\_unit: nm  
notes: "High S/N spectrum from HST STIS; calibrator"

During app startup, read this file and call your Astroquery wrapper to fetch the data if not cached locally. Keep the library small so users can quickly try the app; provide a way for advanced users to add their own entries via UI.

### 4.3 Expose query and retrieval parameters in the UI

Add a settings panel for users to adjust query parameters such as ROW\_LIMIT, cone\_search\_radius, or mission (e.g., GALEX vs HST). Provide defaults but allow overrides. Show helpful tooltips referencing Astroquery’s rate limits and row‑limit guidance[[7]](https://astroquery.readthedocs.io/en/latest/simbad/simbad.html#:~:text=A%20warning%20about%20query%20rate%C2%B6)[[8]](https://astroquery.readthedocs.io/en/latest/gaia/gaia.html#:~:text=Queries%20return%20a%20limited%20number,default%20behaviour%20set%20this%20appropriately).

### 4.4 Provide clear error messages when a data source fails

When ingestion fails due to missing data or invalid FITS headers, catch the exception and display an informative message. For example, “The selected product does not contain a 1‑D spectrum. Please choose a different observation.” Keep asynchronous download tasks per star and show progress bars. Use Gaia.ROW\_LIMIT and vizier.ROW\_LIMIT to avoid retrieving extremely large tables[[8]](https://astroquery.readthedocs.io/en/latest/gaia/gaia.html#:~:text=Queries%20return%20a%20limited%20number,default%20behaviour%20set%20this%20appropriately)[[9]](https://astroquery.readthedocs.io/en/latest/vizier/vizier.html#:~:text=,s).

## 5. Handling custom user data uploads

Allow users to upload their own spectra (FITS, CSV, ASCII). Using the unified I/O interface:

1. **File reading:** call Table.read(fileobj, format=fmt) or NDDataRef.read() for FITS images. For ASCII or CSV, let users specify column names and units; use the PyArrow CSV reader for large CSVs, which can be 15× faster[[13]](https://docs.astropy.org/en/latest/io/unified_table_text.html#:~:text=The%20read,and%20the%20AAS%20MRT%20format).
2. **Unit parsing:** ask the user for the wavelength and flux units if not present in the file. Convert them to Quantity objects and apply to the data.
3. **Metadata merging:** store uploaded spectra in the same internal structure as remote data (e.g., QTable with wavelength, flux, uncertainty columns). Add a flag to indicate user‑supplied data so you can list them separately in the library tab.
4. **Validation:** verify that the file contains a monotonic wavelength axis and numeric flux values. Provide warnings if the header indicates 3‑D data (e.g., more axes than expected) and skip unsupported files.

## 6. Address code architecture issues

### 6.1 Stabilise asynchronous ingestion

The Unexpected ingest result error occurs because OverlayIngestResult is defined inside the Streamlit script. When a rerun occurs, the class identity changes, so results from previous asynchronous tasks are no longer instances of the new class. To fix this:

* Move OverlayIngestResult and related classes (OverlayTrace, etc.) into a separate module (e.g., app/models/overlay.py). Import this module in both the main script and the worker thread. This ensures that the class identity remains constant across reruns.
* In \_refresh\_ingest\_jobs(), instead of checking isinstance(result, OverlayIngestResult), check for attribute names (e.g., hasattr(result, 'payload')) or use duck typing. This avoids strict class identity comparisons.
* Clear cached futures on script rerun. When the script starts, cancel outstanding futures from previous runs and requeue downloads if necessary. This prevents stale results from being considered.

### 6.2 Modularise ingestion and overlay logic

The main.py file currently contains ~3400 lines and mixes UI, ingestion, overlay management and plotting. Consider refactoring into modules:

* **ingest.py** — functions to read files, query remote archives and return NDData/QTable objects. Encapsulate Astropy/Astroquery logic here.
* **overlay\_manager.py** — manages a list of OverlayTrace objects and asynchronous tasks. Handles adding/removing traces, sampling data for plotting, etc.
* **ui.py** — defines Streamlit UI components, calls into overlay\_manager and displays plots.

This separation will make debugging easier and allow you to write unit tests for ingestion functions without the Streamlit context.

### 6.3 Robust error handling and logging

Ensure each asynchronous task catches exceptions and returns a structured error message. When plotting, check if the to\_vectors method exists; if not, re‑instantiate the OverlayTrace from stored data. Logging should include the product ID, file name and error description to aid in debugging.

## 7. Conclusion and next steps

By leveraging Astropy’s unified I/O, NDDataRef containers, units and SkyCoord, and Astroquery’s robust query tools, you can significantly improve the reliability and maintainability of your **spectra‑app**. Curating a small set of well‑documented stars and structuring the library definition will make the app more approachable. Refactoring the codebase to separate ingestion, overlay management and UI, and fixing the asynchronous class identity issue will eliminate the Unexpected ingest result errors and make future development easier.

Implementing these recommendations will require some coding changes, but they align with best practices from the Astropy and Astroquery documentation and will provide a stable foundation for expanding the app’s capabilities in the future.

[[1]](https://docs.astropy.org/en/latest/io/overview.html#:~:text=Astropy%20provides%20two%20main%20interfaces,for%20reading%20and%20writing%20data) [[2]](https://docs.astropy.org/en/latest/io/overview.html#:~:text=Use%20,returns%20a%20QTable) Overview of Astropy File I/O — Astropy v7.2.dev608+g26750bf19

<https://docs.astropy.org/en/latest/io/overview.html>

[[3]](https://docs.astropy.org/en/latest/io/unified_table_fits.html#:~:text=FITS) FITS — Astropy v7.2.dev608+g26750bf19

<https://docs.astropy.org/en/latest/io/unified_table_fits.html>

[[4]](https://docs.astropy.org/en/latest/nddata/index.html#:~:text=,%E2%80%99s%20unified%20file%20I%2FO%20interface) N-Dimensional Datasets (astropy.nddata) — Astropy v7.2.dev608+g26750bf19

<https://docs.astropy.org/en/latest/nddata/index.html>

[[5]](https://docs.astropy.org/en/latest/units/index.html#:~:text=Introduction) Units and Quantities (astropy.units) — Astropy v7.2.dev608+g26750bf19

<https://docs.astropy.org/en/latest/units/index.html>

[[6]](https://docs.astropy.org/en/latest/coordinates/index.html#:~:text=The%20best%20way%20to%20start,specified%20with%20the%20string%20name) Astronomical Coordinate Systems (astropy.coordinates) — Astropy v7.2.dev608+g26750bf19

<https://docs.astropy.org/en/latest/coordinates/index.html>

[[7]](https://astroquery.readthedocs.io/en/latest/simbad/simbad.html#:~:text=A%20warning%20about%20query%20rate%C2%B6) SIMBAD Queries (astroquery.simbad) — astroquery v0.4.12.dev222

<https://astroquery.readthedocs.io/en/latest/simbad/simbad.html>

[[8]](https://astroquery.readthedocs.io/en/latest/gaia/gaia.html#:~:text=Queries%20return%20a%20limited%20number,default%20behaviour%20set%20this%20appropriately) Gaia TAP+ (astroquery.gaia) — astroquery v0.4.12.dev222

<https://astroquery.readthedocs.io/en/latest/gaia/gaia.html>

[[9]](https://astroquery.readthedocs.io/en/latest/vizier/vizier.html#:~:text=,s) [[12]](https://astroquery.readthedocs.io/en/latest/vizier/vizier.html) VizieR Queries (astroquery.vizier) — astroquery v0.4.12.dev222

<https://astroquery.readthedocs.io/en/latest/vizier/vizier.html>

[[10]](https://physics.nist.gov/PhysRefData/ASD/Html/help.html#:~:text=Our%20basic%20policy%20is%20to,spectra%20are%20queried%20in%20ASD) [[11]](https://physics.nist.gov/PhysRefData/ASD/Html/help.html#:~:text=II,NIST%20published%20and%20ASD%20values) NIST Atomic Spectra Database - Help File

<https://physics.nist.gov/PhysRefData/ASD/Html/help.html>

[[13]](https://docs.astropy.org/en/latest/io/unified_table_text.html#:~:text=The%20read,and%20the%20AAS%20MRT%20format) Text (CSV, fixed-width, HTML, and specialized) — Astropy v7.2.dev608+g26750bf19

<https://docs.astropy.org/en/latest/io/unified_table_text.html>