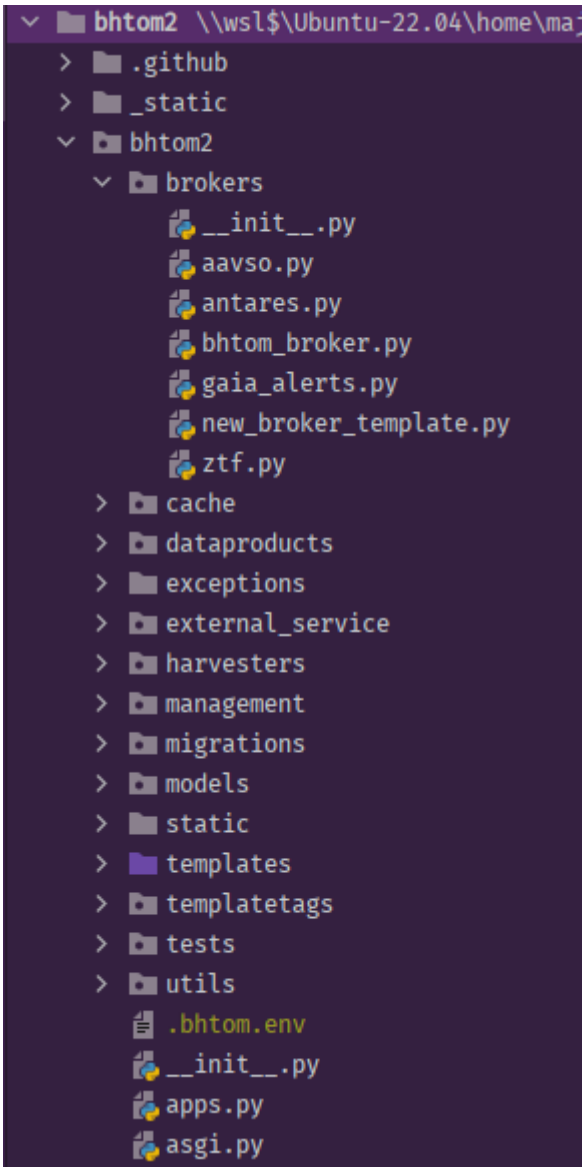
BHTOM2: new broker tutorial

What is a broker?

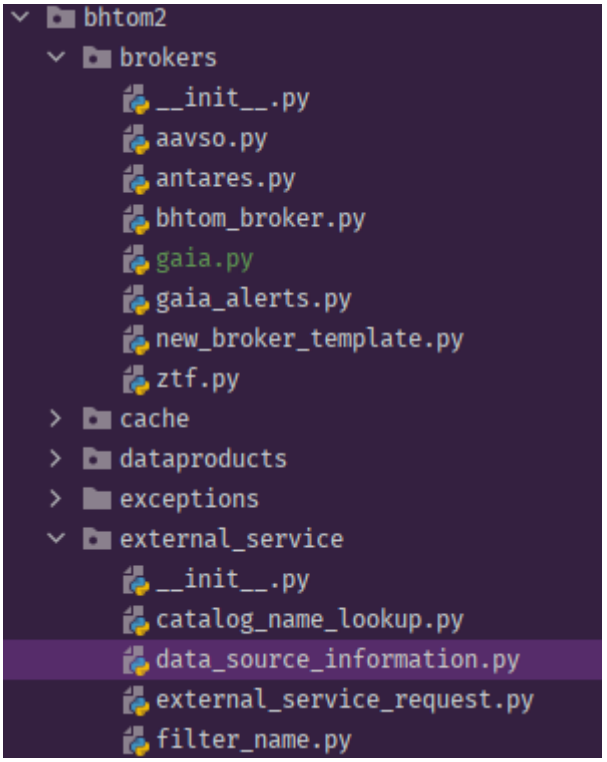
Module for fetching the data from sources like Gaia Alerts, astroquery etc.
There are some existing brokers already. They can be found in bhtom_base/bhtom_alerts (from TOM Toolkit) and bhtom2/brokers (added by us).



The new_broker_template is meant to be a guideline for adding the broker. Let's add the broker for Gaia DR3 as an example.

We will be using Astroquery's API for that, although Gaia TAP+ has also a nice access using pyvo (a Python package for handling ADQL): https://astroquery.readthedocs.io/en/latest/gaia/gaia.html

DataSource
Information about all data sources are in the data_source_information.py file.
This can prevent typos etc. from errorenous data fetches (e.g. by assinging "g(Gaia)" and "g(gaia)" as filters. This way, handling everything on python Enums and treating the names as constants, a consistent convention can be kept).



Begin with adding a new line. Use the upper case (this is a convention for Python constants).
The auto() assings a next possible integer to the Enum value – Enum objects are actually stored as numbers under the hood

```
class DataSource(Enum):
    GAIA = auto()
    ZTF = auto()
    CPCS = auto()
    AAVSO = auto()
    TNS = auto()
    ASASSN = auto()
    ANTARES = auto()
    GAIA_DR2 = auto()
    ZTF_DR8 = auto()
    GAIA_DR3 = auto()
```

Add a new entry to the PRETTY_SURVE_NAME dictionary. This is just for the interface to look nice… (It's used in templates etc.)

```python
PRETTY_SURVEY_NAME: Dict[DataSource, str] = {
    DataSource.GAIA: "Gaia",
    "GAIA": "Gaia",
    DataSource.ZTF: "ZTF",
    DataSource.CPCS: "CPCS",
    DataSource.AAVSO: "AAVSO",
    DataSource.TNS: "TNS",
    DataSource.ASASSN: "ASAS-SN",
    DataSource.ANTARES: "ANTARES",
    DataSource.GAIA_DR2: "Gaia DR2",
    DataSource.ZTF_DR8: "ZTF DR8",
    DataSource.GAIA_DR3: "Gaia DR3"
}
```

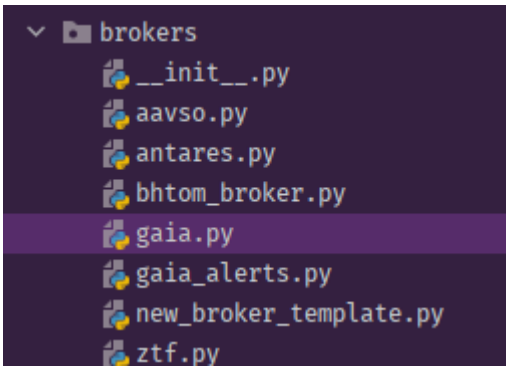
TARGET_NAME_KEYS stores the labels for Target creation forms. Again, just interface purposes.

```python
TARGET_NAME_KEYS: Dict[DataSource, str] = {
    DataSource.GAIA: "Gaia name",
    DataSource.ZTF: "ZTF name",
    DataSource.CPCS: "CPCS name",
    DataSource.AAVSO: "AAVSO name",
    DataSource.TNS: "TNS name",
    DataSource.ASASSN: "ASASSN name",
    DataSource.ANTARES: "ANTARES name",
    DataSource.GAIA_DR2: "Gaia DR2 id",
    DataSource.ZTF_DR8: "ZTF DR8 id",
    DataSource.GAIA_DR3: "Gaia DR3 id"
}
```

(Optional) Add accepted filters. This can be used to validate the incoming data (whether there is no surprise filter, or gibberish data). But it's up to you.

```python
FILTERS: Dict[DataSource, List[str]] = {
    DataSource.GAIA: ["g"],
    DataSource.ZTF: ["g", "i", "r"],
    DataSource.AAVSO: ["V", "I", "R"],
    DataSource.ANTARES: ["R", "g"],
    DataSource.ZTF_DR8: ["g", "i", "r"],
    DataSource.GAIA_DR3: ["rp", "g", "bp"]
}
```

Copy the new_broker_template.py file:



You are probbably not going to use the QueryForm, but this has to be provided to the Broker class. This is for brokers fetching new alerts such as Lasair or MARS.

```python
class GaiaQueryForm(GenericQueryForm):
    target_name = forms.CharField(required=False)

    def clean(self):
        super().clean()
```

Change the names...

```python
class GaiaBroker(BHTOMBroker):
    name = DataSource.GAIA_DR3

    form = GaiaQueryForm
```

Fill in the constructor values.
Gaia is a space observatory, so we treat the facility and observer as constant and pass that as “Gaia”. This can change depending on the data, e.g. for AAVSO.
Update cadence is set to None, because for a data release we just want to check it once.

```python
def __init__(self):
    super().__init__(DataSource.GAIA_DR3)  # Add the DataSource here

    # If the survey is e.g. a space survey, fill the facility and observer names in and treat is as a constant
    self.__FACILITY_NAME: str = "Gaia"
    self.__OBSERVER_NAME: str = "Gaia"

    self.__target_name_key: str = TARGET_NAME_KEYS[self.__data_source]

    self.__update_cadence = None
    self.__cross_match_max_separation = 0.5 * u.arcsec
```

The function process_reduced_data is the most important one. It fetches the data and saves them as ReducedDatumobjects in the database.

```python
def process_reduced_data(self, target, alert=None) → Optional[LightcurveUpdateReport]:


    # There are two options of querying: either by name or coordinates. This depends on the survey.
    # For the coordinates, a BHTOMBroker value is introduced: self.cross_match_separation
    survey_name: Optional[str] = self.get_target_name(target)

    if not survey_name:
        # Change the error message
        self.logger.debug(f'No <Survey> name for {target.name}')
        return return_for_no_new_points()
```

get_target_name checks for a TargetName object linked with the Target and set with the corresponding source_name (so “Gaia_DR3” in our case). This can be added in the Target creation/update form and then queried by that (probably faster than by coordinates).

But we might want to add querying by coordinates as well, so this is why we set cross_match_max_separation here.

If no DR3 id is present, I am querying via astroquery.
It’s good to always wrap querying etc. in a try/except with a logger, so that nothing gets broken because of some external error (something could go wrong also on the broker side!)

```python
def process_reduced_data(self, target, alert=None) → Optional[LightcurveUpdateReport]:
    survey_name: Optional[str] = self.get_target_name(target)

    if not survey_name:

        coord = SkyCoord(ra=target.ra * u.degree,
                         dec=target.dec * u.degree,
                         frame=ICRS)

        try:
            result = Gaia.query_object_async(coordinate=coord,
                                             width=self.cross_match_max_separation,
                                             height=self.cross_match_max_separation).to_pandas()["solution_id"]
            if len(result) > 0:
                dr3_id = result[0]
        except Exception as e:
            self.logger.error(f'Error when querying Gaia DR3 for {target.name}: {e}')
            return return_for_no_new_points()
```

return_for_no_new_points() is a small util method returning an empty LightcurveUpdateReport object. For now, this just contains the number of correct reduced data, but it might be used for other things in the future.

Using some function I’ve written (not important right now- it returns a Pandas table), I’m downloading the photometric lightcurve

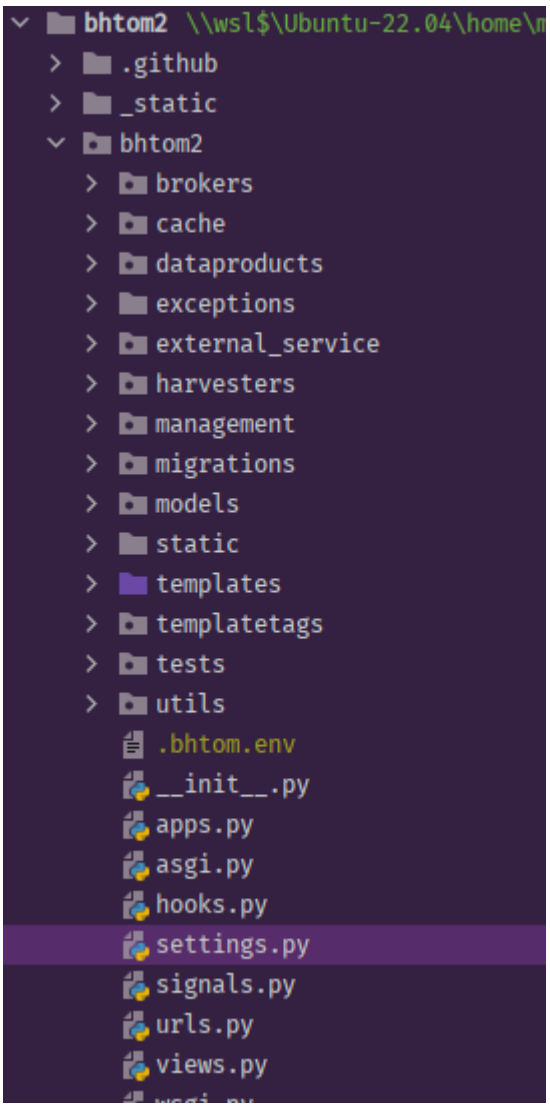
```
self.logger.debug(f'Downloading DR3 lightcurve for DR3 id {dr3_id} (target {target.name}) ... ')
lightcurve: pd.DataFrame = self.download_dr3_lightcurve(dr3_id)

lightcurve['time_bjd'] = gaia_time_to_bjd(lightcurve['time'])
lightcurve['mag_err'] = mag_error(lightcurve['flux_over_error'])
lightcurve['rescaled_mag_err'] = np.sqrt(lightcurve['mag_err'] ** 2 + 0.0045 ** 2)
```

The main loop converts the data to Reduced Datums. Substitute the datum['mag'] etc. so that the fields are correct, depending on the data structure.

!!! IMPORTANT: keep the self.filter_name() around filter so that all filters are in the "filter(survey_name)" format

```
try:
    reduced_datums = [ReducedDatum(target=target,
                                   data_type='photometry',
                                   timestamp=Time(datum['time_bjd'], format='mjd').to_datetime(),
                                   mjd=datum['time_bjd'],
                                   value=datum['mag'],
                                   source_name=self.name,
                                   source_location='Gaia TAP+',
                                   error=datum['mag_err'],
                                   filter=self.filter_name(datum['filter']),
                                   observer=self.__OBSERVER_NAME,
                                   facility=self.__FACILITY_NAME)
                      for _, datum in lightcurve.iterrows()]  # Inline loop
    with transaction.atomic():
        new_points = len(ReducedDatum.objects.bulk_create(reduced_datums, ignore_conflicts=True))
```

Look for the TOM_ALERT_CLASSES variable in settings.py



and add the path to your new broker:

```
TOM_ALERT_CLASSES = [
    'bhtom_base.bhtom_alerts.brokers.alerce.ALeRCEBroker',
    'bhtom_base.bhtom_alerts.brokers.lasair.LasairBroker',
    'bhtom_base.bhtom_alerts.brokers.mars.MARSBroker',
    'bhtom_base.bhtom_alerts.brokers.scimma.SCIMMABroker',
    'bhtom_base.bhtom_alerts.brokers.scout.ScoutBroker',
    'bhtom_base.bhtom_alerts.brokers.tns.TNSBroker',
    'bhtom_base.bhtom_alerts.brokers.fink.FinkBroker',
    'bhtom2.brokers.gaia_alerts.GaiaAlertsBroker',
    'bhtom2.brokers.aavso.AAVSOBroker',
    'bhtom2.brokers.ztf.ZTFBroker',
    'bhtom2.brokers.gaia.GaiaBroker'
    #'bhtom2.brokers.antares.ANTARESBroker',
]
```