Brief introduction

Assume that you're developing an app for an existing Django project. The app should introduce an "Alias" object/model, defined as such:
- "alias" field - string (no specific requirements)
- "target" field - string (a "soft foreign key" to slugs of other models/apps of the existing project; will never be longer than 24 characters)
- "start" field - microsecond precision timestamp/datetime
- "end" field - microsecond precision timestamp/datetime or None

The Alias object is used, unsurprisingly, as an alias, allowing to find other objects (of various different types) from a common entry point. You do not need to care about checking the validity of these other objects (or even care about how many different types of them there will be) or their slug; just accept and return the "target" string as is.
The "start" and "end" fields specify the time range in which the Alias is active. That is, if there's only one Alias for an object, that starts at `2020-01-01 00:00` and ends at `2020-02-01 00:00`, then this means that the object has no Alias before `2020-01-01 00:00` and after `2020-02-01 00:00` , respectively. `start` is inclusive (so the alias is considered available at exactly the `2020-01-01 00:00:00.000000` "start" moment, but not at `2019-12-31 23:59:59.999999`) while `end` is exclusive (so the alias is considered not available at exactly the `2020-02-01 00:00:00.000000` "end" moment, but available at `2020-01-31 23:59:59.999999`).
`end` field set to `None` is a special case: this means "current value", or "continue forever". Most of the time users of the Alias will only care about the current aliases. Historical (with end not None) values will be of interest only in specific cases.
Aliases may overlap (for the same `target`) as long as `alias` value is different. So it's 100% fine to have aliases "useful-object1" and "useful-object2" pointing to the same `target` from `2020-01-01` to `2020-02-01` both.
Aliases may have the same `alias` value (for the same `target`) as long as they do not overlap. So it's 100% fine to have aliases "useful-object" and "useful-object" pointing to the same `target` from `2020-01-01` to `2020-02-01` and from `2020-05-01` to `2020-10-17` .
Aliases MAY NOT overlap with the same `alias` value. It should not be allowed to have aliases "useful-object" and "useful-object" pointing to the same `target` from `2020-01-01` to `2020-02-01` and from `2020-01-31` to `2020-02-05` (note overlap at `01-31`). Even one microsecond overlap is not allowed: two same aliases for same target may never be set in such a way that e.g. first ends at `2019-12-31 23:59:59.543752` and next starts at `2019-12-31 23:59:59.543751` (note that second value is one microsecond less than first). Note how, since `end` is not inclusive, it is still ok to have the first alias end at `2019-12-31 23:59:59.543752` and second start at `2019-12-31 23:59:59.543752`.

Some examples of usage:

```
Alias.objects.create(alias='useful-object', target='types-slug-023xf',
start=timezone.now() - timedelta(days=50), end=None)
alias_obj = Alias.objects.filter(alias='useful-object', end=None)
referred_obj_slug = alias_obj.target
aliases = get_aliases(target='types-slug-023xf', from=<datetime 2020-02-01
00:00:00.000000>, to=<datetime 2020-05-01 05:23:47.657264)
```

This is the main use of an Alias object: to get the slug of a referred object while only knowing it's alias, at a specific point in time. Or to get all Aliases of a specific object (so, with known `target`) in a specific time range.

Your task:

1. Set up a basic Django project with a database of your choice using tools of your choice to do so. This project will contain the single `alias` app.
2. Implement this app (and only it), with:
    a. The `models.py` file containing the Alias object/model definition;
    b. Custom checks and constraints that will ensure validity of all alias records (see description of how Aliases may or may not overlap)
    c. Some means (implementation details up to you) of performing the `get_aliases` example (getting a set of aliases for specific target in the specific time range). The example assumes `get_aliases` is a function. This is not a requirement; you can implement `get_aliases` differently if you think that will work better.
    d. Some means (implementation details up to you) of replacing an existing alias with a new one at a specific time point. That is, something like `alias_replace(existing_alias, replace_at, new_alias_value)` that, when called, will set `end` for the `existing_alias` to `replace_at` moment, create a new Alias with `alias=new_alias_value` and `start=replace_at, end=None`.
    e. Feel free to add any further improvements.

3. Add a test suite covering the `alias` app functionality.
4. Add documentation (docstrings) everywhere you feel is appropriate. Use google-style argument/return value descriptions and type hints, if possible.
5. Follow Python style conventions strictly (PEP-8, PEP-257).
6. It is acceptable to use different style conventions if you specify which you used. If you use other PEP conventions except the two specified, please specify them, too (despite them being "Python style", we may be not familiar with them).
7. **Most importantly, add a README file describing how your project should be launched** (both the development server and the test suite) on a generic Linux system (Ubuntu, Debian, Arch specific instructions also acceptable). **We will not review applications that we are unable to run.**