Deconcentrator

let deconcentrator do the „hard“ work of checking multiple natural language understanding processing providers.

Table Of Content

scope

be a common endpoint for various nlu providers:

- [RASA](https://rasa.com/docs/rasa/)

others aren't implemented yet, but implementation should be trivial:

- [Microsoft LUIS](https://azure.microsoft.com/de-de/services/cognitive-services/language-understanding-intelligent-service/) - [Google Cloud NLU](https://cloud.google.com/natural-language/) - [IBM Watson NLU](https://www.ibm.com/watson/services/natural-language-understanding/)

principles

important frameworks/software pieces

- [nginx](https://nginx.org): reverse proxy (static files) and uwsgi gateway - [uWSGI](https://pypi.org/project/uWSGI/): wsgi implementation - [Django REST framework](https://www.django-rest-framework.org/): REST interfaces, viewsets, generic serialization

```
  logic etc.
```

- [Django](https://www.djangoproject.com/): „The web framework for perfectionists with deadlines.“ - [Celery](https://docs.celeryproject.org/): Distributed task queue for delegating I/O tasks (like doing web requests) - [RabbitMQ](https://www.rabbitmq.com/): async task queue itself - [redis](https://redis.io/): django cache, session cache, celery result backend - [memcached](https://memcached.org/): django cache. - [PostgreSQL](https://www.postgresql.org/): database backend.

important models

- `Method`: the abstraction of a function to retrieve an actual NLU processing result. - `Provider`: the actual provider, which is doing some kind of NLU processing. - `Strategy`: how to select a `Provider` for a specific `Objective` - `Objective`: kind of a task that has to be done. It's the main entry-point, user-supplied. It contains the actual

```
  payload which has to be NLU processed and selects an strategy.
```

- `Job`: the `Strategy` creates jobs from an `Objective`. Each job then has a specific `Provider` to use for processing. - `Result`: the outcome of asking a `Provider`.

implementation details

- `Objective`, `Job` and `Result` make use of non-abc-dispatching (i. e. dispatching without a common abstract base

```
  class). That means:
  - they have a common method with equal signature called `execute()` and
are connected to the same `post_save`
      handler.
  - once an object of one of these classes is `save`d, the `post_save` hook
will call that common method.
  - that method, then, calls the `Strategy` model method for further
handling.
  - to avoid infinite recursion, one has to avoid calling `save()` within
the `Strategy` method, instead using the
      `<Model>.objects.filter(...).update(...)`-approach
```

- the `ViewSet`s design:

1. `Method`s, `Strategy`s, `Job`s and `Result`s can only be read
2. `Provider`s can be CRUD on demand.
3. `Objective`s can be created and retrieved, but not updated or deleted.

how-to

Versioning

Authors