Get behavioural recommendations

All request URLs in the document are relative to the application root. E.g. if the web application is deployed at http://10.0.0.10:8080/api and this document says to request `/recommend/behavioural`, the full URL would be http://10.0.0.10:8080/api/recommend/behavioural

Behavioural recommendations are based on user's profile. They combine knowledge on what the user is interested in with a set of configuration parameters that define weight for each factor.

Let us consider a simple example.

Let us have 2 categories of articles:

- Articles with IDs of **doc-garden-1**, **doc-garden-2**, etc. are related to gardens, garden tools, flowers and so on.
- Articles with IDs of **doc-travel-1**, **doc-travel-2** etc. are related to geographic places. They mostly contain names of such places, transportation terms etc.

In our example articles of the second category are more visited - they have 4 times more reads on average than the garden articles.
Let us have an user with ID of **test-user-1**. They have read 2 articles only: **doc-garden-1** and **doc-garden-3**.

We can get recommended content for this user by calling:

```
GET /recommend/behavioural/?key=<api_key>&userid=test-user-1&count=4&sort=rel
```

This is the result:

```
{
    "version": "Ontotext Recommendations API",
    "type": "RECOMMENDATION",
    "status": "OK",
    "articles": [
        {
            "id": "doc-travel-1",
            "title": "Hello London!",
            "score": 0.6092376112937927,
            "published": "2014-10-24T17:00:37Z",
            "url": "http://example.org/hello-london-article-1",
            "popularity": 12
        },
        {
            "id": "doc-travel-2",
            "title": "Hello London Again!",
            "score": 0.5170392990112305,
            "published": "2014-10-24T17:00:37Z",
            "url": "http://example.org/hello-london-article-2",
            "popularity": 11
        },
        {
            "id": "doc-garden-2",
            "title": "My Spring Garden!",
            "score": 0.47985148429870605,
            "published": "2014-11-01T17:00:37Z",
            "url": "http://example.org/my-garden-2",
            "popularity": 9
        },
        {
            "id": "doc-travel-3",
            "title": "Hello Paris!",
            "score": 0.4229651391506195,
            "published": "2014-10-24T17:00:37Z",
            "url": "http://example.org/hello-paris-article-1",
            "popularity": 11
        }
    ]
}
```


Although the user is interested mostly in gardens, the travel articles are on the top of the list. Higher relevance score is calculated for them. This is due to the fact that they are more popular - that is, they have been visited more often.

The popularity weight may be lowered by POST-ing the request and adding JSON-encoded weight parameter like this:

```
{ "beh.weight.popularity": 0.2 }
```

`Content-type` of the request should be `application/json`. See Advanced recommendation parameters for more information

This would change weight of popularity to 0.2 (down from the default 0.9) and the result would be:

```
{
    "version": "Ontotext Recommendations API",
    "type": "RECOMMENDATION",
    "status": "OK",
    "articles": [
        {
            "id": "doc-garden-2",
            "title": "My Spring Garden!",
            "score": 0.9111301898956299,
            "published": "2014-11-01T17:00:37Z",
            "url": "http://example.org/my-garden-2",
            "popularity": 9
        },
        {
            "id": "doc-garden-4",
            "title": "Work in the Garden!",
            "score": 0.7094958424568176,
            "published": "2014-11-01T17:00:37Z",
            "url": "http://example.org/my-garden-4",
            "popularity": 8
        },
        {
            "id": "doc-travel-1",
            "title": "Hello London!",
            "score": 0.6092376112937927,
            "published": "2014-10-24T17:00:37Z",
            "url": "http://example.org/hello-london-article-1",
            "popularity": 12
        },
        {
            "id": "doc-travel-2",
            "title": "Hello London Again!",
            "score": 0.5170392990112305,
            "published": "2014-10-24T17:00:37Z",
            "url": "http://example.org/hello-london-article-2",
            "popularity": 11
        }
    ]
}
```


Note how relevance scores have changed when popularity is not that important for relevance score.

We can also combine user's profile with what content they are currently looking at. Scenarios like this could be:

- User is reading a news article and we want to recommend them more articles to read.
- User may be reviewing a CV and we may want to recommend suitable job descriptions

The way to add this contextual information is via the **contentid** parameter:

```
GET
/recommend/behavioural/?key=<api_key>&userid=test-user-1&count=4&sort=rel&contentid=doc-
```

Now, as both user's profile and related content is about gardens, we get:

```
{
    "version": "Ontotext Recommendations API",
    "type": "RECOMMENDATION",
    "status": "OK",
    "articles": [
        {
            "id": "doc-garden-2",
            "title": "My Spring Garden!",
            "score": 0.8721830248832703,
            "published": "2014-11-01T17:00:37Z",
            "url": "http://example.org/my-garden-2",
            "popularity": 11
        },
        {
            "id": "doc-garden-4",
            "title": "Work in the Garden!",
            "score": 0.2629146873950958,
            "published": "2014-11-01T17:00:37Z",
            "url": "http://example.org/my-garden-4",
            "popularity": 8
        }
    ]
}
```


Parameters

`/recommend/behavioural` has the following query parameters:

- **userid** (required) - the user to return recommendations for
- **contentid** (optional) - existing article identifier. If passed, recommendations will be computed with respect to this article and the user history
- **count** (optional, default = 10) - maximum number of articles to return
- **sort** (optional, default = rel) - sorting method, valid arguments are "pop"(popularity), "rel"(relevancy) and "date"
- **recency** (optional, default = <empty>) - limit the age of articles returned by the recommendation. Supported values for recency are "amonthago", "aweekago", and integers representing the maximum number of days to go back. For example recency=5 will only return articles newer than five days ago. If omitted, no age filter is applied