Data Query

Your task is to implement a HTTP-based API that will allow web applications to store and retrieve data. Python should be the primary language of your implementation. The data schema is defined as follows:

Fields	Example

`id` - text
`title` - text
`content` - text
`views` - integer
`timestamp` - integer
	`{`
	`  "id": "first-post",`
	`  "title": "My First Post",`
	`  "content": "Hello World!",`
	`  "views": 1,`
	`  "timestamp": 1555832341`
	`}`

API requirements (for the sake of easier reading) are described in other pages.

Evaluation

The solution is expected to implement all requirements that are part of this document. Anything that is not mentioned is up to you and can be implemented in a way that seems most appropriate. Additionally, we expect that solution:

- Works correctly (according to the specification).
- Has a test suite. We will pay attention to coverage, structuring, flexibility.
- Values simplicity; no over-engineering.
- Is maintainable (by you and by others).
- Has an expressive, extendable and testable design.

Uncommon and interesting solutions are great as long as they follow the same key principles listed above.

Submitting

1. Archive project directory (source, build scripts, whatever else seems appropriate) as ZIP.
2. Upload to Google Drive or any other similar service.
3. Send us the download link.

Do not send email attachments because it might get filtered out on the way to our mailboxes!
Do not make public repositories. Thank you.

API

The API consists of two end-points - one to store data and one to retrieve it.

Endpoint	Example
`GET /store?query=...` Takes query as input and returns matching entries. Query format is defined below.	`GET /store?query=EQUAL(id,"abc")`
	`200 OK` `[ { "id": "abc", "title": "Alphabet", "content": "A, B, C, ...", "views": 1, "timestamp": 1555832341 } ]`

Endpoint	Example
`POST /store` Take entity and stores it. ID must remain unique. If record with given ID already exists, it should be overwritten.	`POST /store` `{ "id": "first-post", "title": "My First Post", "content": "Hello World!", "views": 1, "timestamp": 1555832341 }`
	`200 OK` `{}`

Query

The query parameter is a string defining a filter to be applied to the data set. It consists of a couple pre-defined operators, some of which can be combined (see examples).

Operator	Example

`EQUAL(property,value)`
 	
Filters only values which have matching property value.
	`EQUAL(id,"first-post")`
	`EQUAL(views,100)`

`AND(a,b)` Filters only values for which both `a` and `b` are true.	`AND(EQUAL(id,"first-post"),EQUAL(views,100))`
`OR(a,b)` Filters only values for which either `a` or `b` is true (or both).	`OR(EQUAL(id,"first-post"),EQUAL(id,"second-post"))`
`NOT(a)` Filters only values for which `a` is false.	`NOT(EQUAL(id,"first-post"))`
`GREATER_THAN(property,value)` Filters only values for which property is greater than the given value. Valid only for number values.	`GREATER_THAN(views,100)`
`LESS_THAN(property,value)` Filters only values for which property is less than the given value. Valid only for number values.	`LESS_THAN(views,100)`