# Java Developer Test

The purpose of this test is to see how you approach a problem and what your solutions look like. The requirements for this test should be straightforward to grasp. When implementing a solution please keep things simple but well engineered.

# Task Content

Implement an API query and transform this data into report available via REST API. Create a Java web application that provides **/report** service handling PUT, GET and DELETE requests.

- PUT on **/report/{report_id} -** generates report of *report_id* and saves it in database table.
    - If report of given *report_id* does not exist, then new report is created and stored in database. Otherwise, existing report is updated.
    - Successfull PUT request should return 204 and no data.
    - PUT request body JSON - *query criteria*:

```
{
	“query_criteria_character_phrase”: “CHARACTER_PHRASE”,
	“query_criteria_planet_name”: “PLANET_NAME”
}
```

- DELETE
    - on **/report/{report_id}** - deletes report of *report_id* from database.
    - on **/report** - deletes all reports from database
- GET
    - on **/report** returns all report data as JSON:

```
[{

	“report_id”: “{report_id},
	“query_criteria_character_phrase”: “CHARACTER_PHRASE”,
	“query_criteria_planet_name”: “PLANET_NAME”,
	“result”: [{
		“film_id”: “FILM_ID”,
		“film_name”: “FILM_NAME”,
		“character_id”: “CHARACTER_ID”,
		“character_name”: “CHARACTER_NAME”,
		“planet_id”: “PLANET_ID”,
		“planet_name”: “PLANET_NAME”
	},...]
},...]
```

    - on **/report/{report_id}** returns *report_id* data as JSON:

```
{

	“report_id”: “{report_id},
	“query_criteria_character_phrase”: “CHARACTER_PHRASE”,
	“query_criteria_planet_name”: “PLANET_NAME”,
	“result”: [{
		“film_id”: “FILM_ID”,
```

```
            "film_name": "FILM_NAME",
            "character_id": "CHARACTER_ID",
            "character_name": "CHARACTER_NAME",
            "planet_id": "PLANET_ID",
            "planet_name": "PLANET_NAME"
    },...]
    }
```

How report is generated?
The application takes *query criteria* and queries following services:

- https://swapi.co/api/films/
- https://swapi.co/api/people/
- https://swapi.co/api/planets/

to obtain list of films in which appeared characters who contains given *CHARACTER_PHRASE* in their name and whose homeworld planet is *PLANET_NAME*.

The application queries API with user input and stores transformed result in database.

# Technical requirements

1. Java 8 or higher.
2. Maven or Gradle for building application.
3. You may use any java library eg.: guava.
4. Hibernate with in memory database.
5. Spring, eg.: DI

# Verification criteria

1. Does it run.
2. Unit tests run in building cycle.
3. Error handling.
4. Validity and esthetic of querying the data.
5. Validity and esthetic of writing REST API.
6. Performance.
7. Application of software design patterns.
8. Application of Clean Code SOLID principles.
9. Abstraction layers division.

Send us a fat *.war with all dependencies. Share your source code via GitHub.