**Challenge Assignment**

**Congratulations. We like what we see in you, and think you are someone who may succeed at Unity. Before selecting onsite interviews, we’d like to get a little more insight into how you think and how you figure things out.**

Details are below, and cover a bit of programming, a bit of written communication, and no trick questions. Have fun, and good luck.

For the first step, you are going to build a [REST based webservice](https://www.google.com/search?q=rest+web+services&oq=rest+web+services&gs_l=serp.3..0i71k1l8.0.0.0.6298.0.0.0.0.0.0.0.0..0.0....0...1..64.serp..0.0.0.-hWvAvi3Zro). This test project has three important parts which you will be required to complete. You can chose to use any language/framework as you want. Ex. python with flask or java spark or jersey.

**Guidelines**

1. Submit your project using the link below. You can submit via github by pasting the link in the "note section" or attach a zip file.
2. Project should include a readme file with all the steps mentioned about how to run your service and tests.
3. You should also mention if any additional library is needed to be installed or if a specific run time environment is used.
4. The code should be well formatted and have comments for better understanding.
5. You are free to email at [jayt@unity3d.com](mailto:jayt@unity3d.com) if you are confused about any part or have questions regarding it.

**Note -** A good web service also logs every request and response it handles. Write the service in such a way that you also create logs.

**Step 1**

Implement a REST based webservice with following details -

* Implement a REST webservice running on a specific port.
* This service should handle http GET/POST request.
* Service should have a **createProject** api to create a project and save it in a text file called projects.txt. This api should accept only POST request with following parameters..

Example:

**Request -**

Url: http:\\${service-url}:${port-number}\createproject

Header: content-type:application/json

Method: POST

Data :

{

"id": 1,

"projectName": "test project number 1",

"creationDate": "05112017 00:00:00",

"expiryDate ": "05202017 00:00:00",

“enabled”: True

"targetCountries": ["USA", "CANADA", "MEXICO", "BRAZIL"],

"projectCost": 5.5,

"projectUrl": "http://www.unity3d.com",

"targetKeys": [{

"number": 25,

"keyword": "movie"

},

{

"number": 30,

"keyword": "sports"

}]

}

**Response -** 200 ok with response should be “campaign is successfully created” or in case of an error it should respond back with correct error response.

* User should be able to call **createProject** api with post request. Your service should store this data in a text file. A simple example of this saved text file will look like:

Projects.txt

{"id":1,"projectName":"test project number 1","creationDate":"05112017 00:00:00","expiryDate ":"05202017 00:00:00","targetCountries":["USA","CANADA","MEXICO","BRAZIL"],"projectCost":5.5,"projectUrl":"http://www.unity3d.com","targetKeys":[{"number":25,"keyword":"movie"},{"number":30,"keyword":"sports"}]}

{"id":2,"projectName":"test project number 2","creationDate":"05112017 00:00:00","expiryDate ":"05202017 00:00:00","targetCountries":["USA","CANADA"],"projectCost":2.5,"projectUrl":"http://www.google.com","targetKeys":[{"number":25,"keyword":"movie"},{"number":30,"keyword":"sports"}]}

{"id":3,"projectName":"test project number 3","creationDate":"03132017 00:00:00","expiryDate ":"02202019 00:00:00","targetCountries":["CANADA","ENGLAND", "INDIA"],"projectCost":0.5,"projectUrl":"http://www.unity3d.com","targetKeys":[{"number":20,"keyword":"cars"}]}

{"id":4,"projectName":"game project 4","creationDate":"01292017 00:00:00","expiryDate ":"09222017 00:00:00","targetCountries":["IRELAND", "FINLAND"],"projectCost":15.5,"projectUrl":"http://www.unity3d.com","targetKeys":[{"number":11,"keyword":"games"},{"number":10,"keyword":"mobile"}]}

**Step 2**

In this section you should implement an algorithm for selection of project, based on the parameters and rules mentioned below. Web service should have an api named **requestProject** and include a selection algorithm.. An example of request is given below

Example:

**Request:**

Url: http:\\${service-url}:${port-number}\requestproject?

projectid=1&country=usa&number=29&keyword=sports

Header: content-type:application/json

Method: GET

**Response:** 200 ok with following project response if matching project found

Response data -

{

"projectName":"test project number 1",

"projectCost": 5.5,

"projectUrl": "http://www.unity3d.com"

}

Or should return following (in case no matching project found)

{

"message":"no project found"

}

List of url parameters:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name of url param** | **Values** | **optional** | **comments** |
| projectid | <integer> ex. 1 | yes | If sent in request then should always return the project with matching id |
| country | <string> ex. “usa” | yes | If a matching project is found with country then should be returned. |
| number | <integer> ex. 25 | yes | Should select matching project from targetedKeys with number being minimum. |
| keyword | <string> ex. “cars” | yes | Should select matching project from targetedKeys |

**Rules algorithm - Try to implement as many as of following rules in your project.**

* Service should accept get request with above mentioned parameters.
* Service should return projects from file (projects.txt) created in part A.
* Service should never return a project which is expired (if today’s date is above expiry date then project should not be selected).
* Service should always return projects which are enabled, which means the project where “enabled”:true
* Service should never return a project if projectUrl is null.
* Service should return a project with highest price if no url parameter is sent in request. Example url - http:\\localhost:5000\requestProject
* Service should always return the matching project id if it is sent in request. For ex. If request is http:\\localhost:5000\requestProject?projectid=1&country=usa&number=29 then it should return a project with matching id regardless of any other rule.
* If project id is not sent in request url parameters then it should select all the matching projects based on url parameters and return the one with highest cost.
* Url parameters should be considered as AND operator and not OR. Ex. if country=usa&number=30 then it should select all project which has USA in their targetCountries and number is 30 or above in their targetedKeys number.
* Service should return {“message”:”no project found”} if any of the parameter is not matched.

Checkout following examples in the table below for better understanding of these rules.

**Example rules and expected behavior**

|  |  |  |
| --- | --- | --- |
| **Url param** | **Selection of project** | **Final response** |
| No url param | Should select all projects | Return the project with highest projectCost |
| projectid=1 | Select the project with id 1 | If found then return project else return no project found message |
| projectid=5&country=brazil&number=20 | Select the project with id 5 | If found then return project else return no project found message |
| country=brazil | Select all the projects which has brazil in targeted countries | Should return project with highest cost in selected list of projects. As long as project is enabled and not expired. |
| country=usa&number=20 | Select all the project where country is usa and number is >=20 | Return project with highest cost out of selected ones. |
| country=ireland&number=15&keyword=sports | Select all the project where targeted country is ireland, number is >=15 and keyword is sports | Return project with highest cost out of selected ones. If any of the url param is not matched then should return no project found message |

**Step 3**

In this section you should write a test script to test these apis which you have written.

* Create a test script/function where you send a large number of **createProject** requests with different type of test data to create projects.
* Create a test script/function to send invalid data in **createProject** requests and make sure that your service handles invalid data or request
* Create a number of valid projects and make sure that the projects are stored correctly in the text file.
* Write test script/function to send a variety of **requestProject** calls and verify that the correct project is returned. Vary the values in url param and test that your rule algorithm works as expected.
* Write test to verify negative scenarios E.g. if url parameters are wrong, if url is wrong etc. Be creative.
* Verify that every request and response is logged properly.

**Note:** Feel free to use any test framework like junit, testng, pytest for step 3. Having a simple script with asserts is also fine. You can also use postman to test your service and share the script or you can write your own shell script and use curls in it.