16CS352: Cloud Computing – Assignment 2

SelfieLessActs on AmazonAWS (Introduction)

During this semester for the cloud computing course, you will develop a cloud based web application called ***SelfieLessActs*****,** that is used to share information about anything that is good for the society that you observe. Examples of such acts could be

- Picking up a piece of garbage and dumping it in a garbage can
- Road getting laid in your area
- Someone helping a blind man cross the road.
- You helping your mother at home in the kitchen.

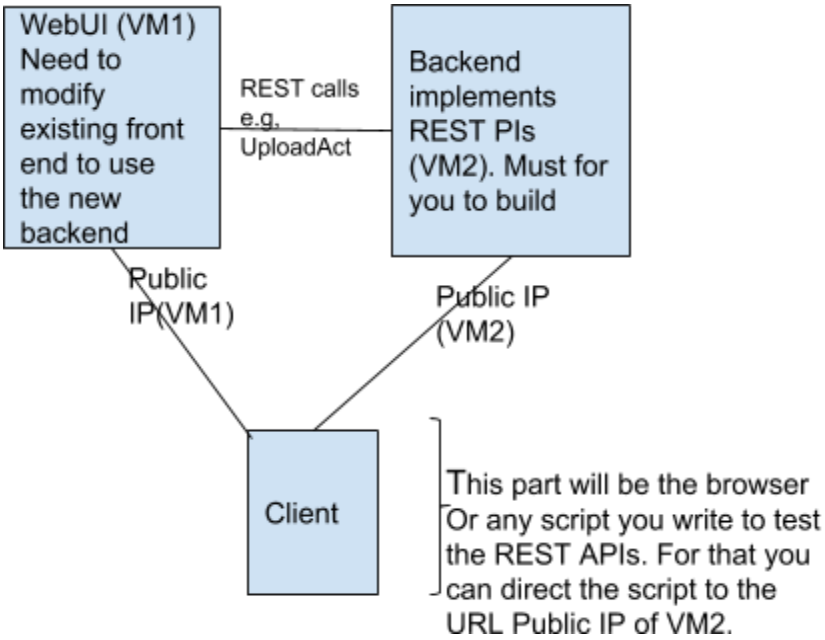
The *SelfieLessActs* application will allow users of the application to upload image of the act with a small caption and categories. A user of the application will be presented with a screen that

- Shows them lists of categories on which Acts have been shared. An act is a combination of an image and a caption for that image.
- Allows them to select to a topic.
- On selection, they will be shown all Acts in a category sorted in reverse chronological order (latest image first).
- Upvote a particular Act.
- Upload an Act.
- Delete an Act

The entire application will be built using Amazon Web Services. We will provide you with an AWS educate starter account preloaded with $75 of credit.

Scope of Assignment 2

For this assignment, you now create now complete the backend processing of SelfieLessActs using a REST API on the AWS instance. The front end has already been created by you. Now you will have the front end use the following specification to create the SelfieLessAcs backend as illustrated in the Figure below.



When creating items on the AWS instance, you need not necessarily need to store it on a database. It is enough to store this somehow on the filesystem in whatever format you want.

Deliverables

1. Each one of the APIs given below must be implemented with proper status codes
2. These APIs must be exposed on the public IP address, so that we can run a test script to test for correctness of the functioning of the API. Hence it is important that you must stick to the specification. Major focus of the credits must be on this.
3. Front end must be modified to use the REST APIs.
4. Please submit a one page report based on the template

Marks: 5
Due Date: Feb 11, 2019

SelfieLessActs: REST API specification

- For each API endpoint, the Response body (sent by you) must be in the corresponding JSON format

- For each API endpoint, the Request body (sent by testing suite) will be in the corresponding JSON format
- From the list of relevant HTTP response codes, you must decide which one to send for a given request body. All response codes given for each endpoint will be tested against by the suite. Response codes are for both success and failure cases.
- You must implement the APIs using the given endpoints.
- Note that below, `{}` represents a JSON associate map and `[]` represents a JSON array.
- Do not assume a max size for any array, string, number. Use appropriate data structures in your backend for this. We may test against your APIs with extremely long strings/numbers.
- **This is the second version of the spec. Please note the changes in the routes and request bodies. The testing script will check against this new spec.**

1. **Add user**
   Route: `/api/v1/users`
   HTTP Request Method: `POST`
   Relevant HTTP Response Codes: `201, 400, 405`
   Request: `{`
   ```
   username: "userName",
   password: "3d725109c7e7c0bfb9d709836735b56d943d263f"
   ```
   `}`
   Response: `{}`
   Comments:
   1) The username in the request body must be unique (case-sensitive), otherwise send the appropriate response code from the given list.
   2) The password field must be a SHA1 hash hex (40 chars long, hex digits only, case-insensitive), otherwise send the appropriate response code from the given list.
2. **Remove user**
   Route: `/api/v1/users/{username}`
   HTTP Request Method: `DELETE`
   Relevant HTTP Response Codes: `200, 400, 405`
   Request: `[]`
   Response: `{}`
   Comments:
   1) `username` in the route must exist, otherwise send the appropriate response code from the given list.
   Example:
   A call to `/api/v1/users/xyz` should delete user "xyz".
3. **List all categories**

Route: `/api/v1/categories`
HTTP Request Method: `GET`
Relevant HTTP Response Codes: `200, 204, 405`
Request: `{}`
Response:

```
{
      "category1Name": 200, // number of acts in category
      "category2Name": 150,
      ...
}
```

Comments:

1) Each category name in the response body must be unique (case-sensitive).

4. **Add a category**

Route: `/api/v1/categories`
HTTP Request Method: `POST`
Relevant HTTP Response Codes: `201, 400, 405`
Request:

```
[
      "categoryName"
]
```

Response: `{}`
Comments:

1) The category in the request body must be unique (case-sensitive), otherwise send the appropriate response code from the given list.

5. **Remove a category**

Route: `/api/v1/categories/{categoryName}`
HTTP Request Method: `DELETE`
Relevant HTTP Response Codes: `200, 400, 405`
Request: `[]`
Response: `{}`
Comments:

1) categoryName in the route must exist, otherwise send the appropriate response code from the given list.

Example:

A call to `/api/v1/categories/xyz` should delete category "xyz".

6. **List acts for a given category (when total #acts is less than 500)**

Route: `/api/v1/categories/{categoryName}/acts`
HTTP Request Method: `GET`
Relevant HTTP Response Codes: `200, 204, 405, 413`
Request: `{}`

Response: `[`

```
    {
        // unique unsigned number
        actId: 1234,

        username: "username",

        // timestamp when act was posted, in given
format
        timestamp: "DD-MM-YYYY:SS-MM-HH",

        caption: "caption text",

        upvotes: 25,

        // base64 string of image binary
        imgB64: "TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvb"
    },
    {
        ...
    },
]
```

Comments:

1) categoryName in the route must exist, otherwise send the appropriate response code from the given list.
2) The `actID` must be unique across all acts in all categories, i.e., globally unique.
3) Timestamp must be in given format.
4) Username must exist.
5) imgB64 must be a base64 string
6) If the number of acts in a given category is larger than 500, return the appropriate response code from the list.

Example:

A call to `/api/v1/categories/xyz/acts` should list all acts in category "xyz".

7. **List number of acts for a given category**

Route: `/api/v1/categories/{categoryName}/acts/size`
HTTP Request Method: `GET`
Relevant HTTP Response Codes: `200, 204, 405`
Request: `{}`
Response: `[`

```
    1031
```

]

Comments:

a. categoryName in the route must exist, otherwise send the appropriate response code from the given list.

Example:

A call to `/api/v1/categories/xyz/acts/size` should give number of acts in category “xyz”.

8. **Return number of acts for a given category in a given range (inclusive)**

Route: `/api/v1/categories/{categoryName}/acts?start={startRange}&end={endRange}`

HTTP Request Method: `GET`

Relevant HTTP Response Codes: `200, 204, 405, 413`

Request: `{}`

Response: Same format as `/api/v1/categories/{categoryName}/acts`

Comments:

1) categoryName in the route must exist, otherwise send the appropriate response code from the given list.
2) Assume all the acts in the category are sorted in reverse chronological order (last to first). Then the range is with respect to that order (1 is for latest act).
3) The given range must be in range (start >= 1 and end <= number of acts in category).
4) The total number of acts requested (end - start + 1) must be less than or equal to 500. Otherwise send the appropriate response code from the list.

Example:

A call to `/api/v1/categories/xyz/acts?start=200&end=400` should list all acts in category “xyz” between index 200 and 400 inclusive (for a reverse chronological ordering of acts in the category).

9. **Upvote an act**

Route: `/api/v1/acts/upvote`

HTTP Request Method: `POST`

Relevant HTTP Response Codes: `200, 400, 405`

Request:
```
[
    1234 // actID
]
```

Response: `[]`

Comments:

1) The `actID` in the request body must exist, otherwise send the appropriate response code from the given list.

10. Remove an act

Route: `/api/v1/acts/{actId}`
HTTP Request Method: `DELETE`
Relevant HTTP Response Codes: `200, 400, 405`
Request: `[]`
Response: `{}`
Comments:

1) The `actID` in the route must exist, otherwise send the appropriate response code from the given list.

Example:

A call to `/api/v1/acts/1234` should remove act with actId “1234”

11. Upload an act

Route: `/api/v1/acts`
HTTP Request Method: `POST`
Relevant HTTP Response Codes: `201, 400, 405`
Request:

```
{
            // unique unsigned number
            actId: 1234,

            username: “username”,

            // timestamp when act was posted, in given
format
            timestamp: “DD-MM-YYYY:SS-MM-HH”,

            caption: “caption text”,

            // base64 string of image binary
            imgB64: “TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvb”
}
```

Response: `{}`
Comments:

1) The `actID` in the request body must be globally unique, otherwise send the appropriate response code from the given list.
2) The timestamp must be in the given format, otherwise send the appropriate response code from the given list.
3) The username must be unique, otherwise send the appropriate response code from the given list.
4) imgB64 must be a base64 string, otherwise send the appropriate response code from the given list.

Additional Resources

1. Flask (http://flask.pocoo.org)
2. Learn more about base64 encoding here - https://code.tutsplus.com/tutorials/base64-encoding-and-decoding-using-python--cms-25588
3. You can manually verify that your APIs work by using Postman (https://www.getpostman.com/).