

Rest Framework – API

Python

Assessment Test

- Write a project to demonstrate blog app api using of Django Rest framework
- Prepare demonstration of Python blog app api under software development principles and follow coding protocols
- The project was built in a MVT concept create an effective interactive interface.
- In this application Perform all operations under function based views
- Create a comment model in the database for the comment application that the Django ORM will manage.
 - Create api which return below data in json format

```
GET /api/
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "title": "First blog post",
        "content": "Hello world!",
        "created_at": "2018-03-12T20:54:39.360653Z",
        "updated_at": "2018-03-12T20:54:39.360681Z"
    },
    {
        "id": 2,
        "title": "Second post",
        "content": "Building APIs with Django is great.",
        "created_at": "2018-03-12T20:54:52.301143Z",
        "updated_at": "2018-03-12T20:54:52.301170Z"
    },
    {
        "id": 3,
        "title": "3rd post",
        "content": "Still making posts here.",
        "created_at": "2018-03-12T20:55:02.620104Z",
        "updated_at": "2018-03-12T20:55:02.620136Z"
    }
]
```

- Make sure all api data return in json format – create separate file of json serializes
- These are a few key options for a REST API request:
- GET — the most common option, returns some data from the API based on the endpoint you visit.

Now go to <http://127.0.0.1:8000/api/1/> and you'll see only the data for the first entry.

The screenshot shows a browser window with the URL <http://127.0.0.1:8000/api/1/>. The page title is "Django REST framework". Below the title, it says "Post Detail". There are two buttons: "OPTIONS" and "GET". Underneath, there is a code block showing the response for a GET request to /api/1/. The response header is "HTTP 200 OK" and the content is a JSON object representing the first blog post:

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": 1,
    "title": "First blog post",
    "content": "Hello world!",
    "created_at": "2018-03-12T20:54:39.360653Z",
    "updated_at": "2018-03-12T20:54:39.360681Z"
}
```

- Cover not just reading/getting content but the full CRUD syntax.

The screenshot shows the Django REST framework's "Post Detail" view. At the top, there's a navigation bar with "Django REST framework" and "WSV". Below it, a breadcrumb navigation shows "Post List / Post Detail". On the right, there are three buttons: "DELETE" (red), "OPTIONS" (blue), and "GET" (blue with a dropdown arrow). A URL input field contains "GET /api/1/". The main content area displays a JSON response for a post:

```
HTTP 200 OK
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "id": 1,
    "title": "First blog post",
    "content": "Hello world!",
    "created_at": "2018-03-12T20:54:39.360653Z",
    "updated_at": "2018-03-12T20:54:39.360681Z"
}
```

Below this, there are two tabs: "Raw data" (selected) and "HTML form". The "HTML form" tab shows a form with fields for "Title" (value: "First blog post") and "Content" (value: "Hello world!"). A "PUT" button is located at the bottom right of the form.

- Admin can able to add and delete records of this system
- Manage proper naming conversion – create appropriate objects name.
- create separate file for all business logics and make them reusable - use modules concepts for implements above logic
- Make sure code prevent from unexpected exception return to the previous screen and accept all details again.
- Make sure Database normalize manage in this project work
- Developer needs to test this product before launching it into the market
- After completion this project upload it on GitHub
 - Upload all features in develop branch after completion all features merge it with main branch

