Developer Note: Implementing API Requests in RcertVault

Step 1: Token Authentication

- Use **TokenAuth (apiKey)** for authentication.
- Prefix: `"Token "`
- Name: `Authorization`
- In: `header`

Step 2: API Documentation

- You can access the default API documentation here:
  http://202.51.70.18/redoc/
- Test api from :
  http://202.51.70.18/doc/

Step 3: Signing

- ❖ **Signing Challenge Request**

  - Make a request to:
  - http://202.51.70.18/api/v1/signing-challenge/
  - Include the parameters as described in the documentation.
  - **Optional Parameter:** `force_create`
    - o If set to `true`, this will delete any existing session and create a **new signing session** for additional PDFs.

- ❖ **Handling the Response**

  - If the user has **auto-signing enabled**, you will receive:
  - {
  -   "signing_session_id": "8b88a1ad-5778-40fc-b216-14296ea2b907",
  -   "max_pdfs": 10,
  -   "expires_in_sec": 50884,
  -   "message": "Existing active signing session returned"
  - }
  - If **auto-signing is not enabled**, you will receive:
  - {
  -   "challenge_id": "57e48e56-efce-42d6-adc8-ead7860c2530",
  -   "challenge_text":
    "eJwNyzkOgCAQAMAvLSCuFJSG2JAoWhuEtcQD6Hi89jNuMnayZhnnbXRrq5lezVu6UiA
    doxQAwAT3p/JEILFHVAf4jtSAR3vpqZQLxf2O5x6umopm0ErW7Jf/VCg+/Q4e9Q==",
  -   "expires_in_sec": 300
  - }

- ❖ **Verify Challenge (If Auto-Signing is Disabled)**

  - Send the `challenge_text` to **DsignerClient** or **EMSigner** as a **form sign**.

- Use the response from DsignerClient/EMSigner in the verify challenge API:
- `http://202.51.70.18/api/v1/verify-challenge`
- Request payload example:
- `{`
- `  "force_create": true,`
- `  "challenge_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",`
- `  "signature": "string"`
- `}`
  - `force_create`: If `true`, creates a new signing session and deletes any existing session.
  - `signature`: The signature obtained from DsignerClient or EMSigner.
- Once verified, the API will return a **`signing_session_id`**, which you can then use to sign PDFs.

❖ **Signing PDFs**

- If you have a **`signing_session_id`**, use the bulk signing API:
- `http://202.51.70.18/api/v1/bulk-sign`
- Include the `signing_session_id` and PDFs you want to sign.

Note:- Session is only valid for 24 hours

Step 4: Verification

❖ **Form Verification**

- Make a request to:
- http://202.51.70.18/api/v1/verify/ verify_text/
- Include the parameters as described in the documentation.
  - **Response:-**

```
{
  "status": "success",
  "message": [
    [
      {
        "cert_name": "Test1",
        "signature_ok": true,
        "hash_ok": true,
        "cert_ok": true
      }
    ],
    true
  ]
}
```
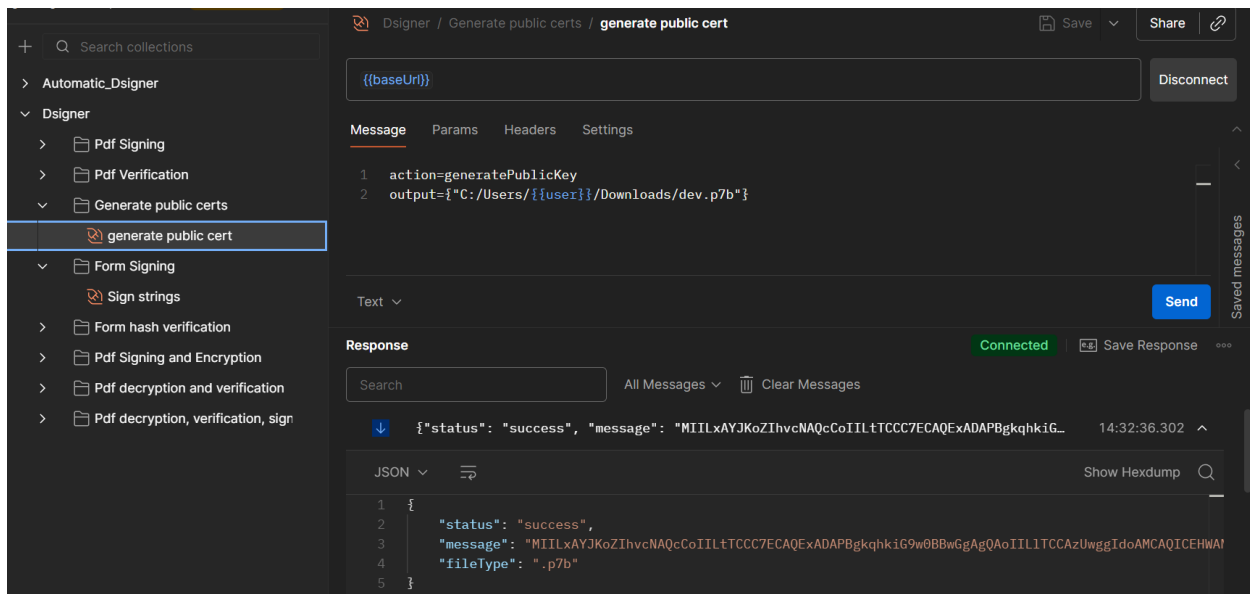
- Here, The final boolean value (true or false) in the response indicates the overall verification status. If it is true, it means that **all checks have passed successfully** — the certificate is valid, the hash matches, and the signature is correct. If it is false, it means one or more of these checks have failed.

❖ **Pdf Verification**

- Make a request to:
- http://202.51.70.18/api/v1/verify/ verify_text/
- Include the parameters as described in the documentation.
- Response is exactly same as the form verification.

Step 5: Creation of the .p7b File for Upload to the Signing Portal

- Use the WebSocket API shown in the image below to generate the .p7b file. The system allows the user to provide a save location as input, which is then used as the output path to store the generated .p7b file on the user's local machine, enabling easy upload to **RcertVault**.
- Alternatively, decode the response message, persist it as a .p7b file, and provide it to the user for download and subsequent upload to **RcertVault**.

Extra Informations:-

top of pdf

Assume,This is one of the page of pdf

Y

(x1,y1,x2,y2)
(400,100,600,200

400   X1

600   X2

Y2

Y1
100   200

bottom

X