BE Engineer take-home challenge

User Accreditation at Yieldstreet

As per SEC regulations, a subset of Yieldstreet's investments are only open to accredited investors. Upon registration, a new user is required to submit documentation to prove their accredited status. Prior to verification by Yieldstreet administrators, the user is only allowed to invest in offerings from the marketplace which do not require verification of accreditation status. Once confirmed, the user is also allowed to invest in offerings whose legal requirements include proof of accredited status.

Challenge

Yieldstreet requires a RESTful HTTP service which (i) is leveraged by administrators to keep track of users' accreditation status, and (ii) serves a client facing web application, serving the current status of a user's accreditation status.

From an administration perspective, two endpoints are required.

i) Accreditation status creation.

```
POST /user/accreditation
Content-Type: application/json

{
  "user_id": "g8NlYJnk7zK9BlB1J2Ebjs0AkhCTpE1V",
  "accreditation_type": "BY_INCOME",
  "document": {
     "name": "2018.pdf",
     "mime_type": "application/pdf",
     "content": "ICAiQC8qIjogWyJzcmMvKiJdCiAgICB9CiAgfQp9Cg=="
   }
}
```

This endpoint is called by an administrator once a user submits a document to prove their status. Given that this document is yet to be verified, internally the accreditation status should be considered as **PENDING**. Valid values for *accreditation_type* are **BY_INCOME** and **BY_NET_WORTH**. If the request is successfully handled, the endpoint must produce a response that looks like this:

```
Content-Type: application/json
```

```
{
  “accreditation_id”: “87bb6030-458e-11ed-b023-039b275a916a”
}
```

i) Accreditation status finalization.

```
PUT /user/accreditation/:accreditationId
Content-Type: application/json

{
   “outcome”: “CONFIRMED”
}
```

The finalization endpoint is called once the administrator has reviewed the submitted document and has finalized their decision, or the accreditation status has expired. Permitted outcome values include **CONFIRMED**, **EXPIRED** or **FAILED**, with this updated state being reflected in the accreditation status identified via the ID provided in the URL. If the request is successfully handled, the endpoint must produce a response that looks like this:

```
Content-Type: application/json

{
  “accreditation_id”: “87bb6030-458e-11ed-b023-039b275a916a”
}
```

From a client facing perspective, one API will be required.

```
GET /user/:userId/accreditation
Content-Type: application/json
```

Returning all accreditation statuses linked to the provided user ID.

```
{
  "user_id": "g8NlYJnk7zK9BlB1J2Ebjs0AkhCTpE1V",
  "accreditation_statuses": {
       “87bb6030-458e-11ed-b023-039b275a916a”: {
              "accreditation_type": "BY_INCOME",
              “status: “FAILED”
       },
       “C031a5da-d59a-4d35-b1f0-0f8324dcc156”: {
              "accreditation_type": "BY_NET_WORTH",
              “status: “CONFIRMED”
       }
  }
}
```

Requirements

1) Validation of request bodies is expected, ensuring a valid JSON object is passed which matches the shape of the provided examples. A sensible non-successful HTTP response should be returned if the request is invalid.
2) There are no persistence requirements for the submitted accreditation statuses; feel free to use any in-memory solution that satisfies the above requirements.
3) The admin endpoints should also perform a set of sensible semantic validations:
    a) A **FAILED** accreditation status should not be updateable further via API
    b) A **CONFIRMED** accreditation status can be **EXPIRED** via API
    c) A user should not be able to submit another accreditation request if there already is a **PENDING** request

    Sensible non successful HTTP responses should be returned in case of failure.
4) A background process should also be in place which automatically expires **CONFIRMED** accreditation statuses whose last update was older than 30 days.
5) The only technical requirement is to use the Java programming language - other than that, feel free to use any frameworks/libraries/tools you're familiar with.
6) You must submit a complete, fully functional solution via a public Git repository (such as Github/BitBucket). A run script for a linux based environment should be included which initializes the service and exposes the endpoints on **http://localhost:9999**
7) Include a README.md file which provides brief answers to the following questions:
    a) Please provide an overview of the architecture, briefly mentioning any applied patterns.
    b) An additional requirement is submitted which requires an audit log of the historical accreditation status updates (beyond the current status of each). Please provide a high-level overview as to how such functionality can be achieved.
    c) The business is doing well, resulting in a multiple fold increase in traffic. As a result the GET endpoint is being hammered, resulting in the service's performance degradation. Please describe high level idea/s how the implemented service can be scaled to handle more traffic.

Evaluation considerations

Your solution will be evaluated on several points. Is this a solution that builds, runs on the expected base URL and successfully accepts HTTP requests? Does it work as outlined in the requirements? How clean is the code? How well is it architected and documented? Have steps been taken to verify the code? Are the README responses satisfactory?

If you move on to the next step of the interview process, some of the questions in the next rounds will be based on your solution - expect to talk about how to scale your solution, potential performance pitfalls, architecture choices, etc.