smart applications. for business.

enigmatry

Development of a New Web API Endpoint for Client Financial Document Retrieval

The goal of this assessment is for you to show us your knowledge and skills as a software developer. You are free to make choices with regards to applying every principle and practice that you would in a real situation so that this solution satisfies all the coding standards.

Objective: Create a new .NET C# Web API endpoint that allows clients to retrieve a financial document within our suite of applications used for submitting tax returns.

All the mentioned services in the task description can work with any data source, such as databases, web APIs, storage, and so on. The candidate is allowed, depending on their available time, to demonstrate the implementation using a technology of their choice or to use mocked services. It's on the candidate to create and define models, relation, configuration, and test data.

Usage of any data access technologies is welcome but is not required. You don't have to use SQL server or EF, you can use whatever is more convenient for you (local files, hardcoded values, app memory…). Focus should be on clean code and best practices.

Acceptance Criteria:

- The completed task must include a fully functional Web API endpoint.
- Demonstrate testing technique on one or more cases.

New endpoint request and response models

Request Format:

```
{  "productCode": "string",  "tenantId": "guid",  "documentId": "guid" }
```

Response Format:

```
{  "data": "serialized and anonymized JSON",                  // check step 6.
  "company": {                                                // check steps 4. and 7.
                    "registrationNumber": "string",
                   "companyType": "string"
                 }
```

Request Flow:

1. **Validate Product Code:**
   Create a service which accepts ProductCode and returns if the product is supported.
   If product code is not supported, exit the flow and return HTTP status 403.
   For example we can have products with ProductCode: ProductA, ProductB, ProductC…,
   We should support at least two products (ProductA and ProductB).

2. **Tenant ID Whitelisting:**
   Create a service which accepts TenantId and returns if Tenant is whitelisted.
   If Tenant is not whitelisted, exit the flow and return HTTP status 403.
3. **Client ID Whitelisting:**
   Create two services:
   - The first service accepts TenantId and DocumentId as inputs and returns the corresponding ClientId and ClientVAT
   - The second service accepts TenantId and ClientId as inputs, validates if the ClientId is whitelisted, and returns true or false.

   Ensure that the ClientId is whitelisted. If it is not whitelisted, terminate the flow and return an HTTP status code of 403."
4. **Fetch Additional Client Information:**
   Create a service which accepts ClientVAT as input and returns the RegistrationNumber (string) and CompanyType (string).
   CompanyType: small, medium, large.
5. **Company Type Check:**
   Check CompanyType, if CompanyType is small exit the flow and return HTTP status 403.
6. **Retrieve Financial Document for Client:**
   Create a service which accepts TenantId and DocumentId and returns a financial document in json format (find an example in this document).
   Financial document json structure depends on ProductCode.
   Structure may vary upon different products and can be more complex.
7. **Enrich Response Model:**
   Add RegistrationNumber and CompanyType to the response.
   Use data retrieved in step 4.
8. **Financial Data Anonymization:**
   Anonymize the data retrieved in step 6., without deserialization. Replace the values of all properties with "#####", except if it's stated in configuration that property should be either hashed or left unchanged. Each product has its own configuration.
   Find an example in this document.
   Field types:
   - a) Hash the value
   - b) Leave the field unchanged

     For example for ProductA all properties should be returned except properties hash: account_number

9. **Return Response:**

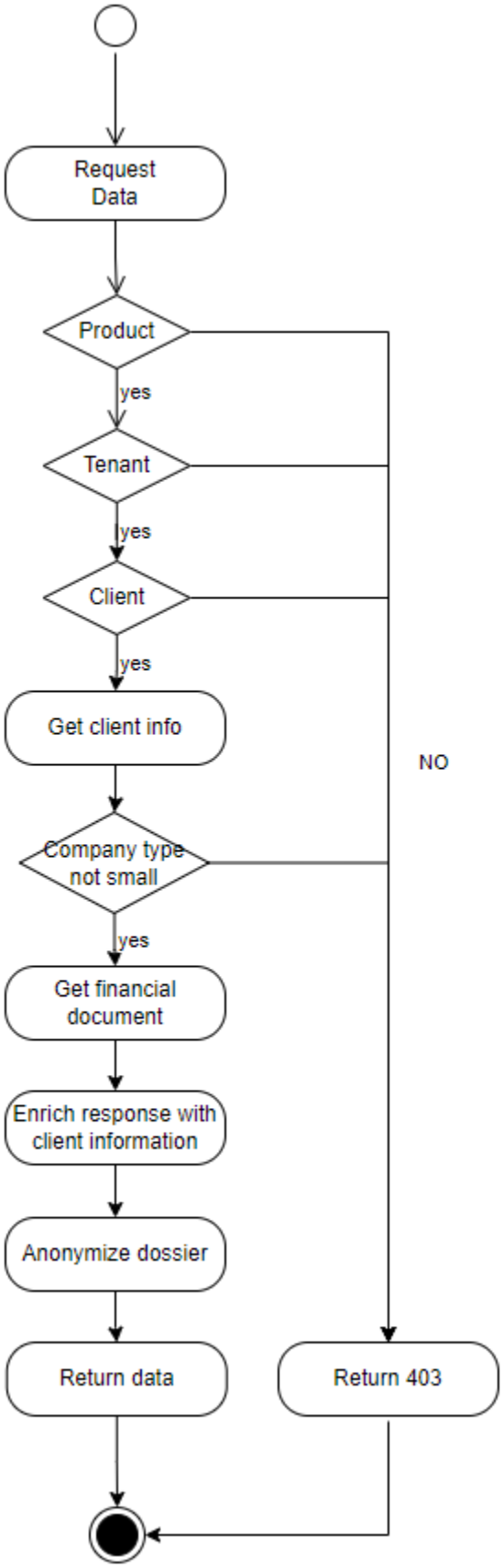
```
{  "data": "serialized and anonymized JSON",
   "company": {
          "registrationNumber": "string",
          "companyType": "string"
         }
}
```

Example of financial document json (step 6.)


```
{
   "account_number": “95867648”,
   "balance": 42331.12,
   "currency": "EUR",
   "transactions": [{
         "transaction_id": “2913”,
         "amount": 166.95,
         "date": "1/4/2015",
         "description": "Grocery shopping",
         "category": "Food & Dining"
      }, {
         "transaction_id": “3882”,
         "amount": 6.58,
         "date": "24/4/2016",
         "description": "Grocery shopping",
         "category": "Food & Dining"
      }, {
         "transaction_id": “1143”,
         "amount": -241.07,
         "date": "25/12/2019",
         "description": "Gas station purchase",
         "category": "Utilities"
      }
   ]
}
```


Example of anonymized financial document json (step 8.)


```
{
   "account_number": ”d3sg4sxos23zxvhads”,     // hashed
   "balance": 42331.12,                        // not changed
   "currency": "EUR",
   "transactions": [{
         "transaction_id": ”#####”,            // masked by default
         "amount": 166.95,
         "date": "1/4/2015",
         "description": "#####”,
         "category": "Food & Dining"
      },
   ]
}
```


Request Data
Product
yes
Tenant
yes
Client
yes
Get client info
Company type not small
yes
Get financial document
Enrich response with client information
Anonymize dossier
Return data
NO
Return 403