Invoice Service

Description

As part of the AREX invoice financing services we receive invoice details from multiple sources within the AREX ecosystem and the format of invoices can vary from between sources.

For that purpose we want to build a centralized component within our architecture that would handle all the invoice management functionalities.

The service will support two types of invoices:

- Accounts Payable (AP) - invoices where our customer is the entity that has to pay for the services/products (debtor);
- Accounts Receivable (AR) - invoices where our customer is the entity that is issuing an invoice (issuer);

On the first iteration we expect the service to track a couple of details regarding the invoice:

customer_id	UUID	ID of AREX's customer
invoice_number	integer	Invoice number
currency	string (ISO 4217 Codes)	Currency of the invoice
face_value	integer	Invoice final value (in cents). E.g., €100.00 will be received as 10000
counterparty_vat(1)	string (EU VAT number)	Vat number of the other party in the invoice;
issue_date	string DD-MM-YYYY	Date when the invoice was issued
due_date	string DD-MM-YYYY	Date when the invoice is due

(1) For AP invoices the **counterparty** is the entity that issues the invoice. For AR invoices the **counterparty** is the entity that is the debtor.

Invoice Creation

The invoices can be delivered to us from multiple sources with different standards so this service needs to be able to handle the different variations and also be easy enough to customize so we can add new behaviours.

XML Invoice

An XML invoice is an invoice that we receive where we extract all the data from an XML file that respects the standard Finvoice 1.3. The XSD for this format can be found here.

From that invoice we extract all the details mentioned above except the **issuer_id** that needs to be provided.

When the invoice is received via this standard we want to store the original file as an invoice preview. More info on *Invoice Preview* below.

Data Invoice

A data invoice is a type of invoice where we receive in the request payload all the information mentioned above.

For this type of invoice we also receive a PDF of the invoice that we use as the invoice preview. More info on *Invoice Preview* below.

Invoice Preview

The invoice representation is just a document that we use as a source to display more details about the invoice received.

For the sake of this challenge let's assume that retrieving the representation of the invoice it will return the content-type and the payload it will be in base64 encoding.

Invoice Attachments

An invoice can have zero or more attachments. These attachments are documents that provide some insight into the context of the invoice i.e., some invoices may have a contract attached to prove that both companies entered a commitment.

Validations

Some validations should apply to the invoices:

- The group (customer_id, invoice_number) must be unique for AR;
- The group (counterparty_vat, invoice_number) must be unique for AP;

Invoice Checking

After any invoice uploaded we will need to run some validations. The first one will be to check if the counterparty vat provided is valid. For that we can use the VIES API and check that the provided VAT is valid and if so store the company name as another field for the invoice.

If this checking fails we need to set the invoice to a failure state and we should be able to update the counterparty vat to fix any issues.

Invoice Ready

The invoice will be marked as ready if it has been checked with success and if an attachment has been provided.

After an invoice has been marked as ready we won't be able to edit its details any more.

Deliverables

- Implement the following service behaviour:
    - Invoice Creation (XML / Data);
        - For the XML invoice a file **invoice.xml** will be provided as a sample;
    - Upload a preview for an invoice;
        - A file named **invoice_preview.pdf** will be provided as a sample;
    - Retrieve invoice preview;
    - Upload attachments for an invoice;
        - A file **invoice_attachment.pdf** will be provided as a sample;
    - List attachments;
    - Retrieve attachment;
- Service endpoints should be implemented using gRPC;
    - For the file endpoints usage of gRPC streaming is advised;
- Add relevant test coverage to the service;
- Provide a way to call the endpoints. It can either be:
    - A simple REST Api that will act as a proxy to your gRPC endpoints and a Postman collection to interact with those endpoints;
    - A set of Go test commands and a README that explain how to use those commands;
- Provide a Docker Compose that will allow us to start the application;
- Push your implementation to a private GitHub repository and share it with the user ealves-pt;

What is expected

- Service implementation in Go using gRPC;

- PostgreSQL as the DB technology;
- Application of async tasks whenever necessary. Extra points if you use AWS SQS;
- Storage of files uploaded can be done in the DB. Extra points if you use AWS S3;
- Clear separation between the different layers of the service;
- Clean and well structured code;
- Complete test coverage for your implementation;