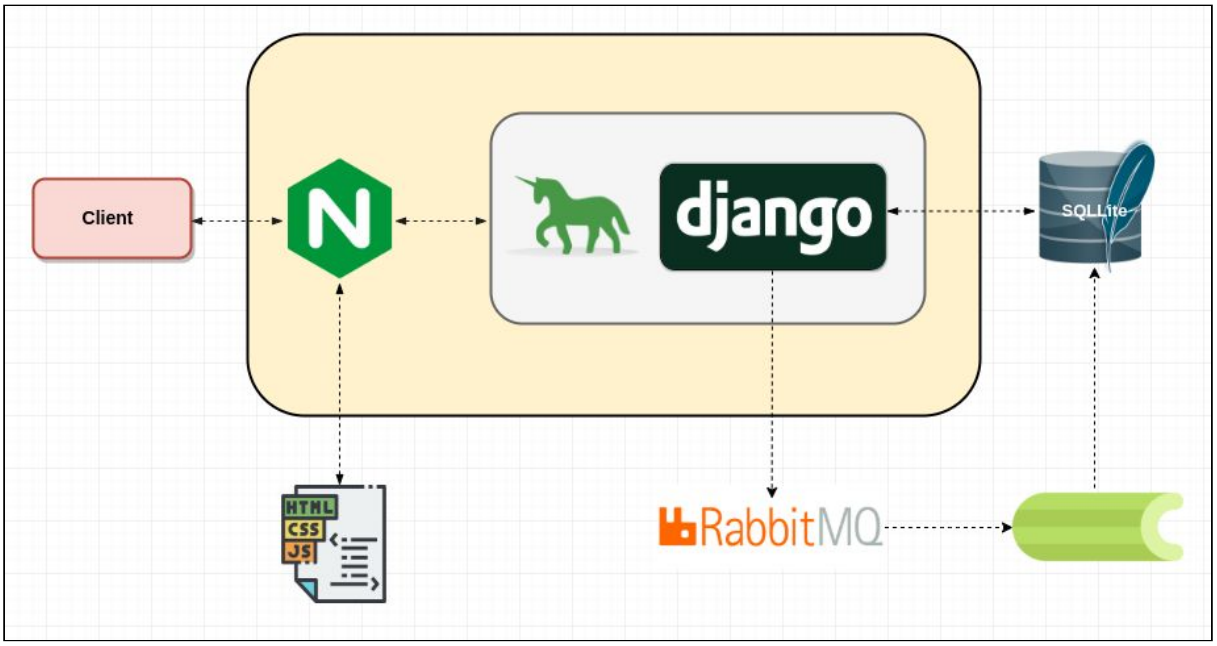
Invoice Management System

Problem Statement:

Design a web application for invoice management with API endpoints for

- ❏ Users to
  - Upload the invoice
  - Check invoice digitization status
  - Get digitized invoice details
- ❏ Staff users to
  - Add or update invoice details
  - Update invoice digitization status

Architecture Diagram:

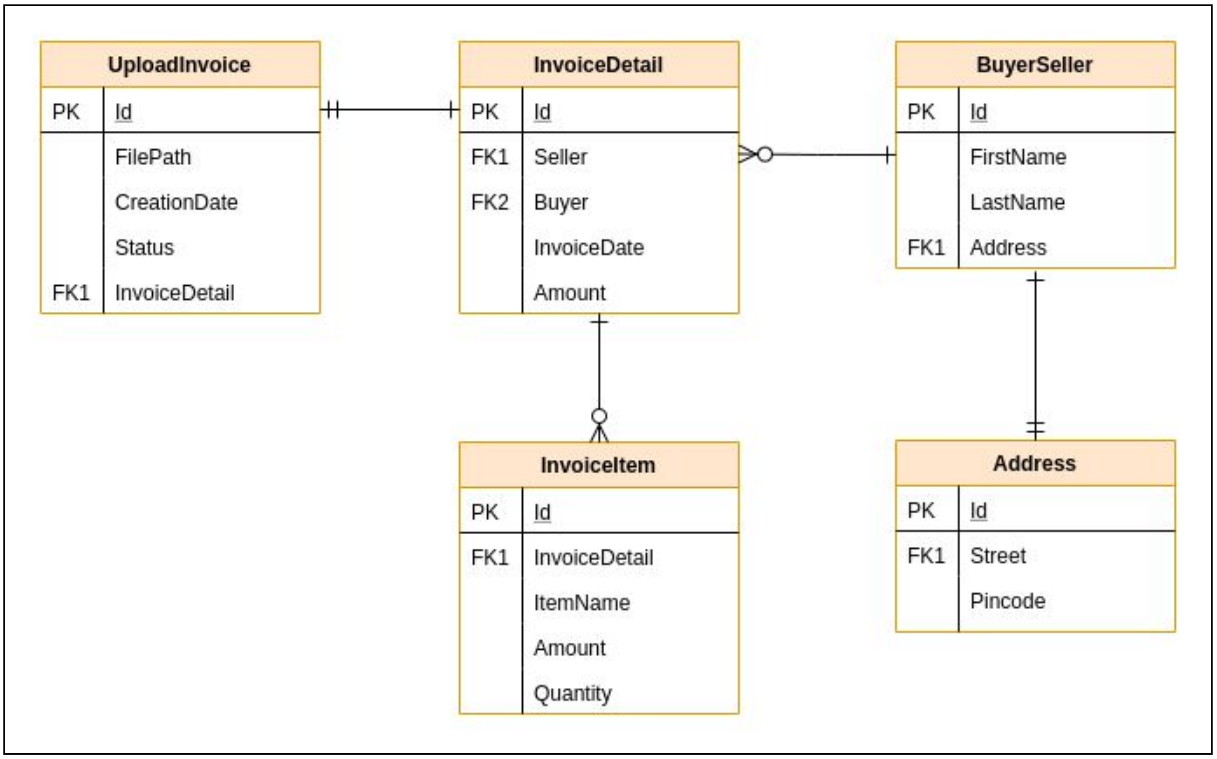


High-level approach:

The user (vendor) can upload the invoice using the dashboard on the front-end side. These invoices will be added to the RabbitMQ queue for asynchronous processing. We'll maintain a status for each of these invoices in the database to keep the user updated about the invoice processing! The status of the invoice when it is in the queue would be *InProgress*.

Celery workers will pick up the processing task from RabbitMQ queue and will process the invoice document to convert it into a structured data (For the purpose of this assignment, this part is a black-box and the assignment only focuses on providing some APIs to upload the invoice and add/modify any of it s fields).

ER Diagram



Here are some of the API endpoints:

To upload the invoice - *api/invoice/upload/*

- Creates an entry in the *UploadInvoice* table with its *status* as *New*
- Sends invoice processing task to the RabbitMQ

Celery workers will pick the task from RabbitMQ, updates the *status* in the *UploadInvoice* table to *InProgress* and processes the invoice asynchronously. Once the invoice is processed, it will create an entry in the *InvoiceDetail* table and update the *status* in *UploadInvoice* table to *Done*. Updation of status of *UploadInvoice* can be done with the help of Google PubSub also.

To check the status - */api/invoice/upload/{upload_id}*

Users can check if the uploaded invoice has been processed using this API and if the invoice has been digitized its corresponding structured format can be accessed using the API */api/invoice/{invoice_id}*

To check the digitized structure - */api/invoice/{invoice_id}*

In case of any discrepancy between the uploaded copy of the invoice and its digitized version, the staff users can update the invoice details using */api/invoice/{invoice_id}* API. They can also create a new entry for the invoice using */api/invoice/* API.

Staff user can update the *status* of the invoice digitization using */api/invoice/upload/{upload_id}* API.
As *invoicedetail* table is normalized, we have to fire 4 queries for inserting or updating data in *address*, *BuyerSeller*, *invoicedetail* and *invoiceitem* table.

API endpoints:

/api/invoice/upload

Purpose: Upload the invoice with the help of AJAX call
Method: POST
Request body:
{
File_path: File to be uploaded
}

/api/invoice/upload/{upload_id}

Purpose: Update the invoice processing status - To be used by Staff users only. Status - 0: New, 1: *InProgress*, 2: Done
Method: PATCH
Request Body:
{
Invoice_id: 123,
Status: 1
}

/api/invoice/

Purpose: Create a new entry for invoice detail. To be used by staff users only
Method: POST
Request Body:

```
[
  {
    "seller": {
      "first_name": "Virat",
      "last_name": "Kohli",
      "address": {
        "street": "Chandni Chowk",
        "pin_code": "421501"
      }
    },
    "buyer": {
      "first_name": "Rohit",
```

```
      "last_name": "Sharma",
      "address": {
        "street": "Thane",
        "pin_code": "421003"
      }
    },
    "amount": 400.0,
    "invoice_number": "123",
    "invoice_item": [
      {
        "name": "Apple",
        "quantity": 1,
        "price": 50.0
      },
      {
        "name": "Mango",
        "quantity": 1,
        "price": 44.0
      }
    ]
  }
]
```

/api/invoice/{invoice_id}

Purpose: Update the invoice details. To be used by staff users only
Method: PATCH
Request Body:

```
[
  {
    "seller": {
      "first_name": "Virat",
      "last_name": "Kohli",
      "address": {
        "street": "Chandni Chowk",
        "pin_code": "421501"
      }
    },
    "amount": 400.0,
    "invoice_item": [
      {
        "id":: 1
        "name": "Apple",
        "quantity": 1,
        "price": 50.0
```

```
        },
      ]
    }
]
```

Assumptions:

- The invoices will have some fixed number of columns
- As for this assignment, I'm skipping the authentication part and also the authorization bit as the application doesn't focus on the users' information at the moment. Although there are APIs for the staff users; the assumption is that these APIs will only be accessed by the staff users. However, it is not difficult to scale this application to add the information of these users. We can create a users microservice and put in some validations to have authorization in place
- BuyerSeller one to one relation with the InvoiceDetail table

Trade Offs:

- Normalized invoice details table, this helps in saving a lot of space but it increases the processing time for each record as we have to fire 4 database queries:
    - First insert into the *address* table and get the *address_id*
    - Update the *buyerseller* table with the *address_id*
    - Update the *invoicedetails* using the buyer and seller id
    - Update the *invoiceitems* using the invoice details id

- Storing the uploaded files' details and the invoice details in separate tables
    - Due to this once a user uploads an invoice, he is given an upload id to check the invoice processing status. When invoice is processed, the user is provided with an invoice id to check the invoice details.
    - Having all the details in one table will provide a better user experience as the user will have to remember only one id