# Document Template Breakdown

1. Create predefined HTML templates in a database and link these to template types.
2. Create APIs for Templates:
   1. Get all templates (Store result in Template type object in FE so don’t need to fetch every time we create a new doc):
      1. *Inputs:*
         1. *OrganisationID: string*
      2. *Output:* 
         1. ***See page 4 of doc***
   2. Get template (Open existing document)
      1. *Inputs:*
         1. *DealDocumentID: string*
         2. *DealID: integer*
         3. *TemplateTypeID: string*
      2. *Outputs:* 
         1. *HTML code of saved document (includes input fields that were saved).*
         2. *Template placeholder data from Deal ID*
   3. Create document template:
      1. *Inputs:* 
         1. *Deal ID*
         2. *Final HTML code*
         3. *PDF Filename – Generated PDF from jsPDF*
         4. *Email Address:*
         5. *Email body info:* 
            1. ***NEED TO DISCUSS DEFAULT BODY***
   4. Update document template\*\*
      1. *Inputs:*
3. Working with Templates in FE:
   1. Use a modal component in Angular (e.g., Bootstrap modal) to display the fetched HTML templates.
   2. Display the selected template on the right-hand side of the modal with matching/corresponding input field names
   3. Provide input fields for each placeholder of PDF on the left-hand side.
   4. Pre-populate input fields (place holders) from deal data as per selected template. User can edit the fields if needed.
   5. Generate PDF:
   6. When the user confirms the input data and clicks a "Generate PDF" button, fill the placeholders in the selected HTML template with the provided data and generate a PDF using jsPDF.
   7. Display generated PDF:
   8. Allow user to confirm PDF data is correct.
   9. User can then “Save PDF” or “Save and Email PDF”.

# Database:

## Tables:

* + DealDocuments:

|  |
| --- |
| * + - ID |
| * + - DealId |
| * + - DocumentTemplateID |
| * + - DocFileName     - HTML Code |
| * + - Status |

* + DocumentTemplates:
    - ID
    - Name
    - Base HTML
    - Status
  + DocumentTemplateInputs
    - ID (incremental counter)
    - DocumentTemplateID
    - FieldID
    - FieldName
  + DealDocumentValues
    - DealDocumentID
    - HTMLCode
    - TemplateInputID
    - InputValue

## Backend:

**Description**:   
Implement document service, DB context, and API endpoints to fetch and save document template data.

##### APIs:

1. **GetDocumentTemplates (OrganisationID: string): JSON Array**
   * **Response:**

{

"DocumentTemplates": [

{

"TemplateID": "",

"TemplateName": "",

"HTMLCode": "",

"InputFields": [

{

"Id": "",

"Name": "",

"Value": ""

},

{

"Id": "",

"Name": "",

"Value": ""

},

{

"Id": "",

"Name": "",

"Value": ""

}

]

}

]

}

# **Frontend (Angular):**

### **View Documents:**

* Create new “Documents” side menu:
  + Contains list of created documents for deals and doc types:
    - Deal No
    - Document
    - Created By
    - Download link?
* Create an Angular service to call the API endpoint and fetch HTML templates
  + **Note**: Fetch all templates when initialising the documents form.

## **Add Document:**

* Design a modal component to display the fetched templates and input fields for placeholders.
* Section 1:
  + Search for deal no
    - Display some deal summary data to confirm correct deal (cannot edit this info)
  + Select document template
* Section 2:
  + Display the selected template on the right-hand side of the modal with matching/corresponding input field names
  + Provide input fields for each placeholder of PDF on the left-hand side.
    - E.g. First name, surname, address
  + Pre-populate input fields (placeholders) from deal data based on selected template
  + User can edit the fields if needed.
* Implement functionality to generate a PDF using jsPDF.
* User can view PDF to confirm data, formatting, etc is correct.
* User can then save PDF, \*\***email PDF\*\*, OR \*\*Save and Email\*\***

**Considerations:**

* Can user send to multiple email addresses
* Do we just want the user to be able to download and then send the PDF themselves?
  + If not, can a user edit and resend?

**Notes**:

* Sanitizing the HTML templates fetched from the database to prevent XSS attacks.
* Validating user input before generating the PDF.