*“Micro front end scalable application to upload and retrieve contract documents”*

Assuming the title to be **Contract Document Management System**. So, I’ll term it further as CDMS.

* The Users who got the privilege for accessing the CDMS will be able to login to the system and land at their Dashboard.
* The Dashboard will show a variety of menu like.
  + People
    - The CDMS will allow the Users to create participants.
    - The CDMS will have an option for creating the roles.
    - User will be able to assign the access privileges/roles to the participants.
    - This way User can co-author a Contract.
  + Categories
    - CDMS will present the user with default contracts available in the system.
    - User will be able to create custom categories, so that in further screens he can assign a Contract to a Template and to a category.
  + Template
    - User will be able to create a Template.
      * Template will have a name
      * Description
      * Background image
    - While Template creation User will have to select the Category from the dropdown of available Category’s.
  + Contracts
    - The landing page of the Contracts menu shows a list of already created Contracts.
    - User will have the ability to search from the list of Contracts by applying certain filter criteria.
    - User will be able to create a Contract by clicking Add new contract button.
    - The Contract creation page has the below fields.
      * Contract Title
      * Contract description
      * Start date
      * End date
      * Template
      * Contract executor
      * Contract executor Id
      * Vendor details
        + Id, Name, More details
      * Vendee ID
        + Id, Name, More details
      * Browse button to select the files, and Upload
    - The landing page of Contract will show a list of available contracts and there will be a Download button with respect to each search result.
      * In further enhancements we may think on providing a checkbox w.r.t each search result and to download multiple Contracts at same time in a zip file.
* At each page and step during while the process we’ll show informative icons which on click we’ll show the step-by-step details in different popups.

**Technical Details**

* UI
  + We’ll use Angular 15 for this development, and we use Jest for writing the tests.
  + We will have different Angular Components created for each of the specified menu item
  + We create all the components using Angular CLI so to make sure we get a standard component and no syntax issues.
  + We will have all the test cases written in the spec files.
  + We will have a service layer defined, and a shared services layer if needed.
    - The service layer will have the Interceptors defined to validate the requests
    - The service layer will have the http communication between the Angular application and the Java Backend REST service
    - We will cover with Units tests for the service layer too.
  + We define our dependencies in the .package file
  + We will write Cucumber feature files and execute the feature files using Cypress automated tool.
  + We will define our configurations in angular json file
  + We will configure our required Coding standards in the .eslintrc json fiie
* Backend
  + We will have different Spring Rest Controllers w.r.t each component
  + We will write all the business logic in Service layer in the Interface implementation classes, we term all these classes as Spring service objects.
  + We’ll have Spring beans integrated with each other using the Spring Auto wiring.
  + The service layers will then invoke the corresponding Repository classes by passing the entity objects.
  + We’ll define our DB connection in the application yaml files.
  + We’ll make sure all the transaction related methods will be supporting Transactional.
  + The contract documents we will store them in Amazon AWS.
  + We will write a Util class for S3 related operations like uploading the document.
    - Using a S3 client
    - PutObjectRequest and provide the filename and make a key out of it, and provide the S3 bucket name. and any applicable capabilities like public-read.
    - Using the S3Client object now put the InputStream.
    - To acknowledge the upload, wait until the file exists using the WaiterResponse object of Amazon AWS.
  + The Util class for downloading the document will perform download from the specified S3 bucket.
    - The GetObjectRequest of Amazon AWS needs a bucket and key to be specified.
    - The downloaded request is then transformed to a InputStream, and then converted to a file output stream which the users can view as a pdf or any other file types.
* Deployments
  + For UI,
    - We will define our deployment configurations in .ecs json file for our Angular applications.
    - We will define the containers desired, containers minimum, containers maximum, and we will define the deployment runtime and mention the AWS account id in the arn. (Amazon resource names)
    - And we will define different deployment environments in the ecs json file.
  + Backend,
    - The application is deployed into AWS ECS services
    - We will define the number of required containers, and the min and max containers to be created so they’ll be able to create automatically depending on the Load
    - We define our configurations in the .ecs json file and the client secrets in the ecs launch file.