Detailed Design AI-Powered Document Validation

JDC Team 3313

Sarah, James, Max, Shreyas, Arjun, Sriman

Client: Andrew Huot (Microsoft)

Repository: https://github.com/Arjunm03/jic3313-aidocumentvalidationservice

Table of Contents

Introduction

Background

This project is an AI-Document Validation Service. The purpose of this project is to improve the workflow of validating compliance documents by introducing an automated solution. Our solution consists of an application that takes in uploaded compliance documents and leverages AI to automatically validate or reject the uploaded document.

Document Summary

The system architecture section describes the design choices in building the underlying architecture of our solution. It describes the static and dynamic elements of our system.

The component design section describes the components of the major application features, the frontend, the backend, and our database, and outlines the relationship between these components.

The data storage design section describes the schema of our data and further explains our choice of database and data storage methods.

The UI section describes the major UI screens the user will be interacting with upon using this application, and why specific design choices were made.

Terminology

Javascript – A programming language
MongoDB – A database service
MVC (Model View Controller) – A design pattern where elements of the application are divided into a model, a view, and a controller which are cyclically updated.
Node js – A backend and frontend framework
PII- Personally Identifiable Information
React js – A frontend framework
UI- User Interface

Detailed Design – System Architecture

Introduction

This section outlines the architecture of the AI-Powered Document Validation Service, a web application designed to automate the process of document information verification using a custom-built machine learning model, built with the Azure OpenAI Studio by Microsoft. Our client is Microsoft's Strategic Missions and Technologies project manager Andrew Huot, who aims to utilize our project to streamline federal government document acceptance and rejection processes, for example for security clearances. Our architecture diagrams model the structural components of the application and the relationships between them. The software is distinguished by five key functional components: the React.js-based front-end for user interaction, the Node.js backend for application logic and also for data mapping, MongoDB for data storage and the custom classification model built upon Azure OpenAI Studio.

Static System Architecture

The static architecture diagram below (Figure 1) displays our modified Model-View-Controller (MVC) architecture, providing a snapshot of the system's components and their interrelations. Our rationale for picking this architecture is three-fold. First, MVC architecture is commonly used for web applications like ours due to the easy integration into JavaScript frameworks and storing and accessing PDF files from the database. Second, MVC allows for the team to develop separate components simultaneously due to the low dependency and the different skillsets needed for the five components shown. Thirdly, modifications to the AI Classifier will not affect the other components due to the abstraction in this architecture, which is necessary as the classifier may be updated and fine-tuned regularly. In this design, we show the structural layout and the dependency relationship using arrows between the components as follows:

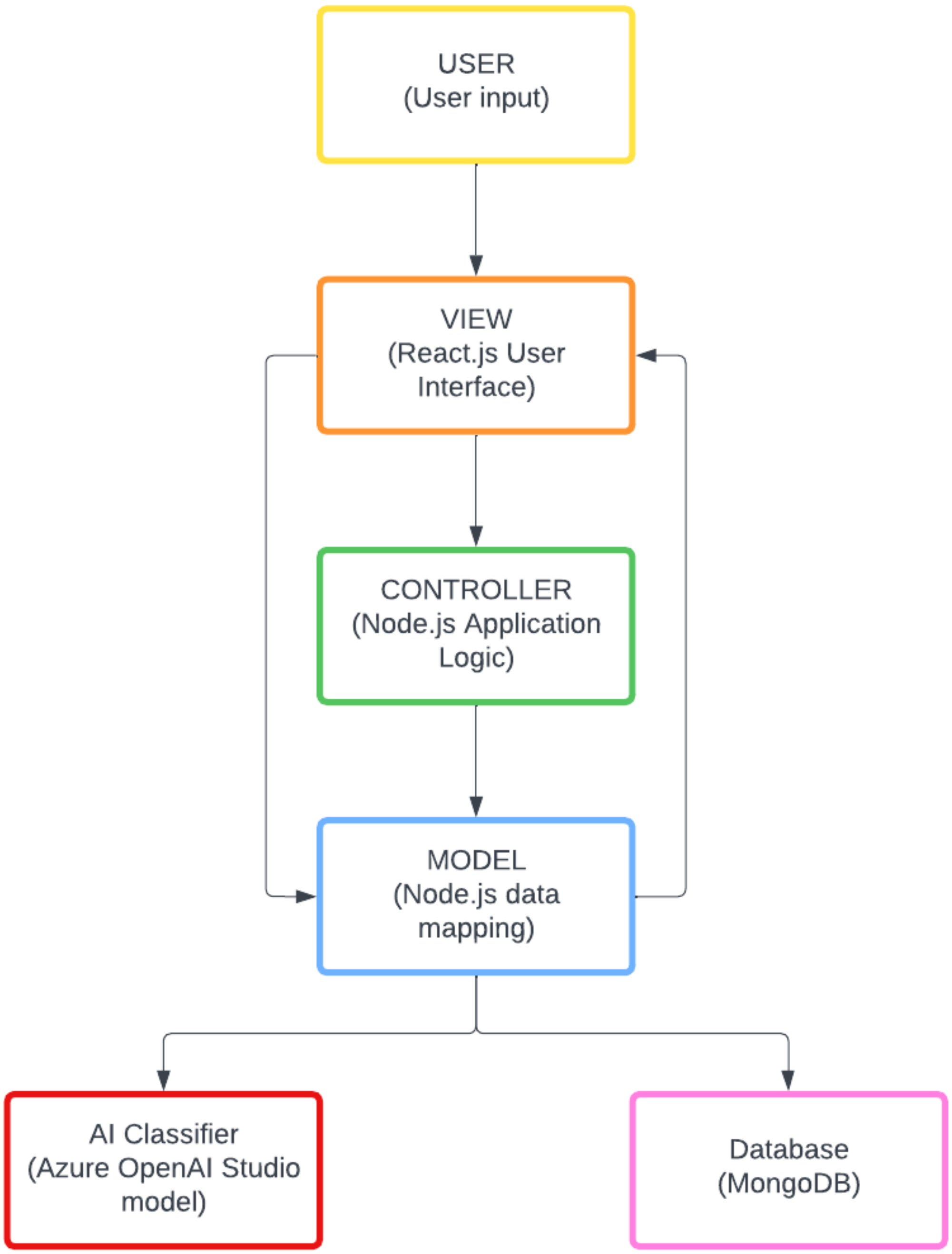


***Figure 1.** Static Architecture Diagram (MVC Architecture).*

In the above figure, the User interacts with the user interface (the View component of the model), and when necessary, the Controller component requests mappings from the Model. These are largely from the Database, with regards to user account information and files, but may also be from the AI Classifier component when the automated AI validation needs to be run on a document.
Our modification is in the final step where the Model also fetches information from the AI Classifier component. When the user decides to run the automated validation service, the Controller and Model

will ask for the results from the AI Classifier and essentially generate new data in the form of classification results. This modification is trivial and doesn't change the overall system architecture (MVC) since it is akin to another source of data as a subset of the Model component itself.
As can be seen in Figure 1, the model is nearly acyclic in terms of dependency as the View depends on the Controller, the Controller depends on the Model and the Model depends on the Database and the AI Classifier. However, in cases where the page rendering does not need controller logic but needs value data, the View communicates directly with the Model for object-relational mapping and updates itself. In the following section, the colors of each component shown in this section will be used for the associated components shown in the Dynamic System Architecture diagram.

Dynamic System Architecture

The dynamic architecture diagram below (Figure 2) demonstrates the runtime behavior of the system as it processes a document validation request. This sequential diagram details the interactions between the user, the front-end user interface, the back-end controller logic, and the external Azure OpenAI Studio model during a typical document validation scenario.
The user story displayed is "**As a verifier, I want to automatically validate the document so I can get a preliminary validation without manual work.**" The diagram below follows the steps from user login, through document upload and specification, to the final delivery of validation results. The solid arrows here represent user actions or layer requests, while the dashed arrows represent returned results or messages:

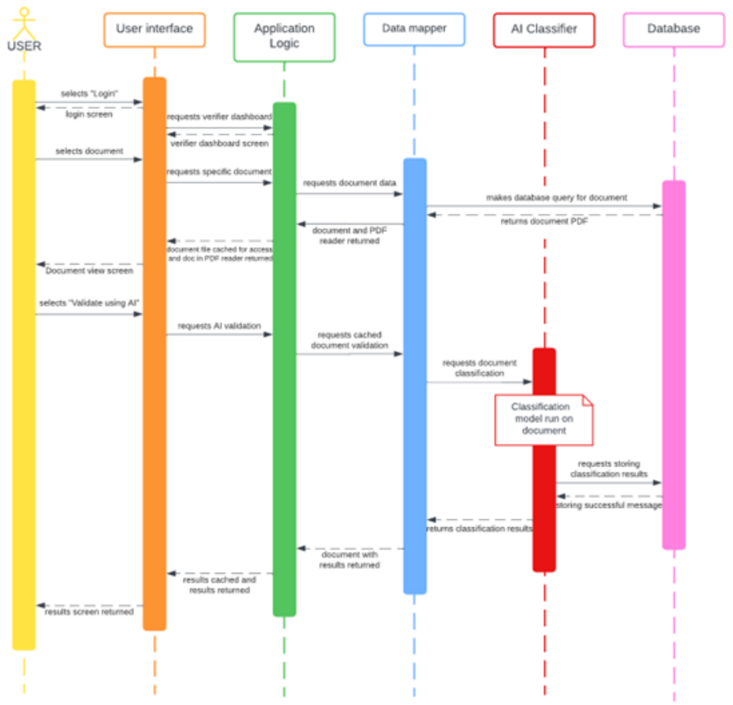

USER
User interface
Application Logic
Data mapper
AI Classifier
Database
selects "Login"
login screen
requests verifier dashboard
verifier dashboard screen
selects document
requests specific document
requests document data
makes database query for document
returns document PDF
document and PDF reader returned
document file cached for access and doc in PDF reader returned
Document view screen
selects "Validate using AI"
requests AI validation
requests cached document validation
requests document classification
Classification model run on document
requests storing classification results
storing successful message
returns classification results
document with results returned
results cached and results returned
results screen returned

Figure 2. Dynamic System Architecture displaying a "Running AI Validation" user story.

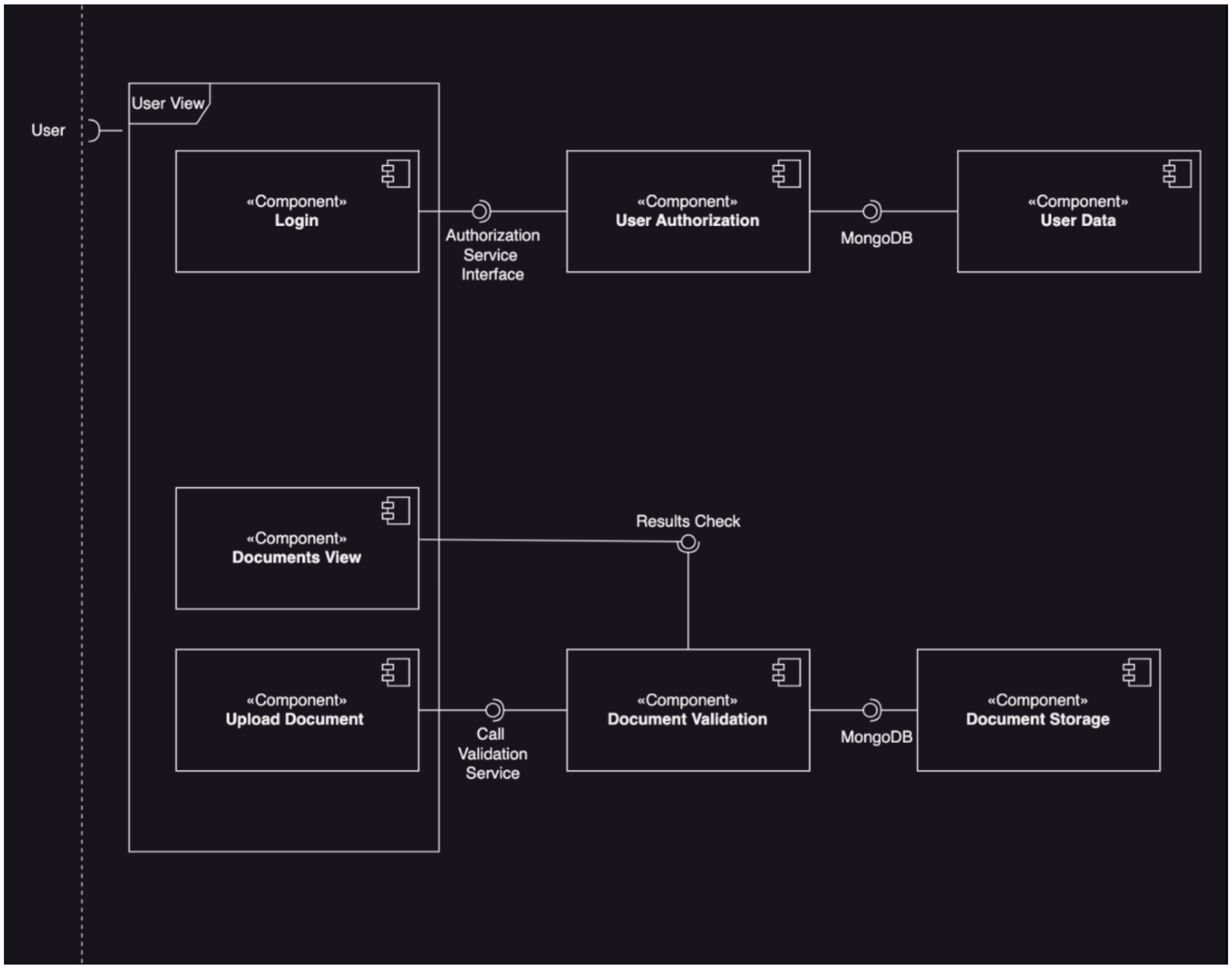
The figure above displays the user and the five key components, and shows a login and two large requests within the user story. The first is fetching a requested document and displaying it, and the second is running an AI validation request and returning its results. Following Figure 2, we can observe that the user interface largely defers these requests to the application logic and model, where a few optimizations take place - such as caching the displayed document and later, its results. We also observe the storing of the classification results to the database once they have been generated. These optimizations are made so that classification can be run quickly without making another database query and results are stored immediately.

Detailed Design- Component Design

Introduction

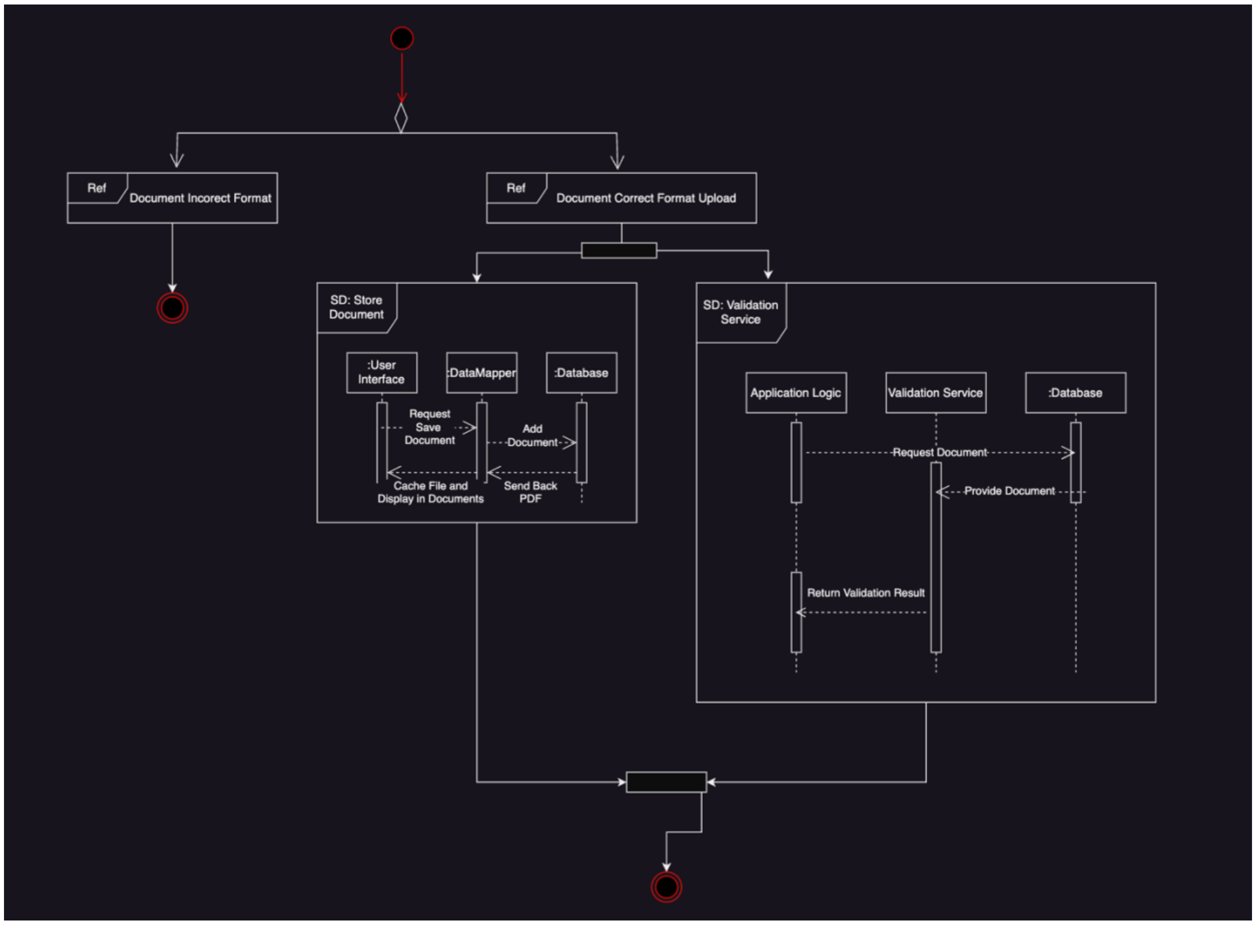
This section will describe the design of the basic components of the AI-Powered Document Validation Service. These include the major functional components of the authorization and login workflow and the major functional components of the document validation and storage functionalities. These are spread across the layers of the application service, with backend, frontend, AI service, and database service included.

Static Component View


User
User View
«Component» Login
Authorization Service Interface
«Component» User Authorization
MongoDB
«Component» User Data
Results Check
«Component» Documents View
«Component» Upload Document
Call Validation Service
«Component» Document Validation
MongoDB
«Component» Document Storage

This component diagram shows the necessary components and interfaces for the authorization and validation services. MongoDB is used to store both user data and documents and is the primary interface. The authorization service interface and call validation service interface are both API services. The document validation component is used in document validation, and also saving a document to the document storage. In addition, the results interface is the necessary backend logic in order to correctly display validation results to the frontend.

Dynamic Component View



This Interaction Overview Diagram shows the results of a user uploading a document and pressing the validate button. Initially, backend logic tests whether the document is in the correct format. If so, we move on to storing the document and using the AI validation service to validate the document. Both processes are described in SDs which elaborate the necessary steps between the objects of the application. Validation results are provided to the backend application logic and then the process terminates.

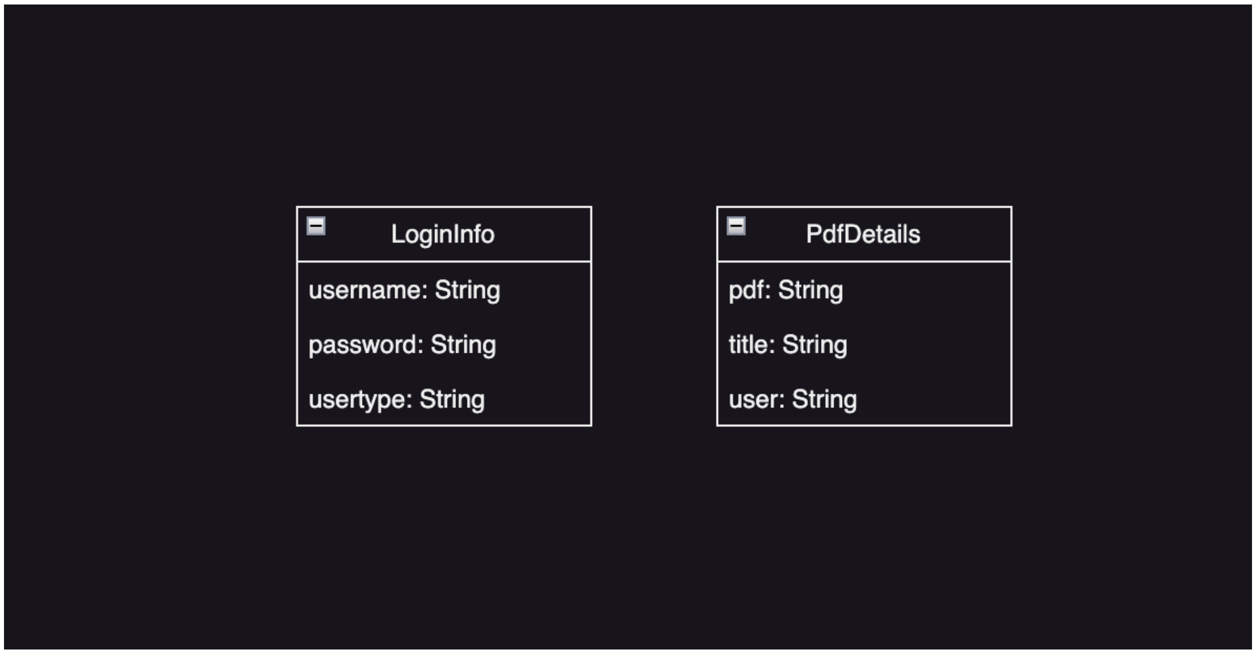
Detailed Design-Data Storage Design

Introduction

This section outlines the design for the data storage component of the AI-Powered Document Validation service. Two major data entities are stored in this process: the data pertaining to users and the document's data. MongoDB is used as the database service for this application. User data is saved upon login and when authorization occurs. Documents are saved whenever a document is uploaded and a call to validate is made.

Database Use

In order to store user data and the documents that are uploaded, MongoDB was chosen as a database service. The following diagram depicts the schemas used. All data will be stored as Strings.



File Use

Files will be stored as PDFs's with the PDF path stored in the database as a string. The actual documents are stored as PDF files and not in MongoDB because they are quite large files.

Data Exchange

Security

Since this service will be integrated into an internal Microsoft product, most of the security concerns will be settled by the final integration of the product. Eventually, the user info and document details will be stored in Azure and appropriately encrypted, especially given that the user info and the documents do contain PII. Authorization will be handled by the API in the backend, and the username and password will be encrypted.

Detailed Design UI-Design

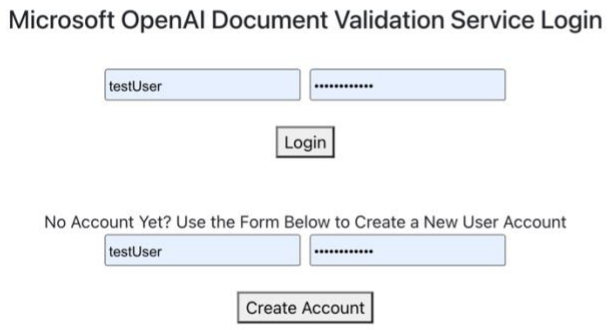
Intro

Our UI was designed with functionality and clarity in mind. The application should ideally be very simple and easy to use, with only a few actions that the user is required to do, so we chose a minimalistic design with one major screen for all options, in line with Nielsen's heuristic of aesthetic and minimalist design. We kept a blue theme throughout the design because the project is for a Microsoft internal tool, and blue is one of Microsoft's major colors.

Major Screens

Login

The login is the first screen the user sees when they log into the application. The screen is simple and prompts the user to either log in or the create an account as a new user. Both options will then create a dialog box popup letting the user know that they have successfully created an account, or if they have successfully logged in, or used the wrong password.


Microsoft OpenAI Document Validation Service Login

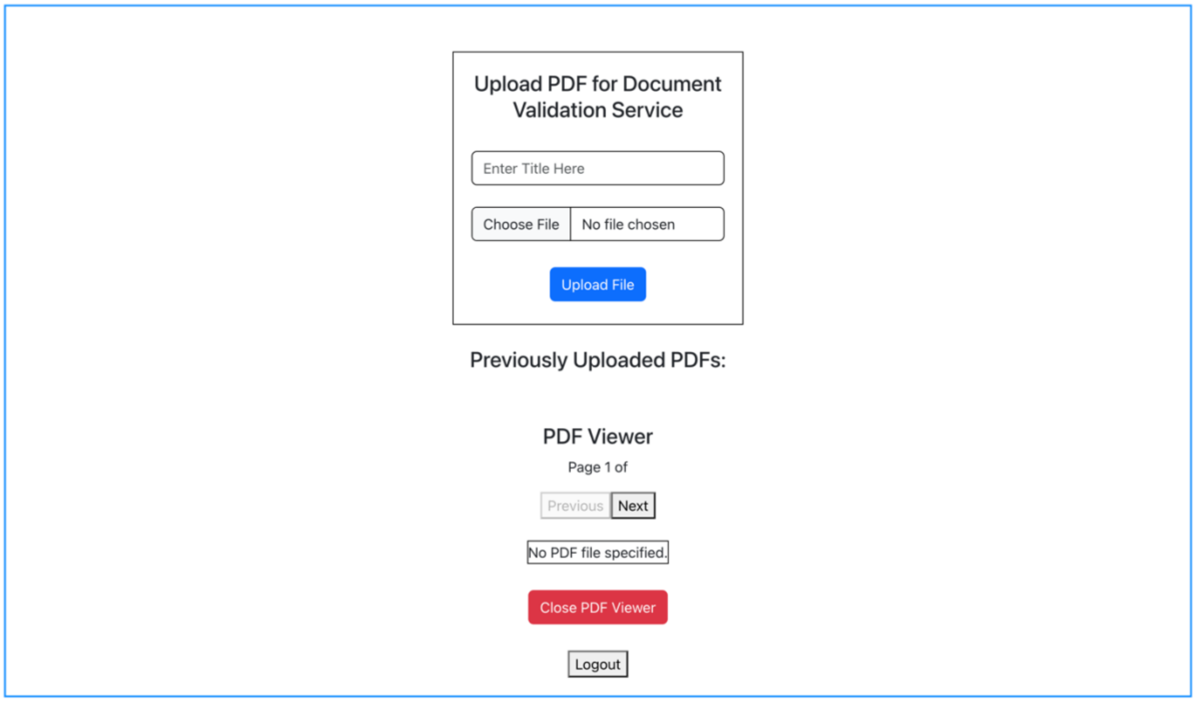
testUser ••••••••••••

No Account Yet? Use the Form Below to Create a New User Account

testUser ••••••••••••

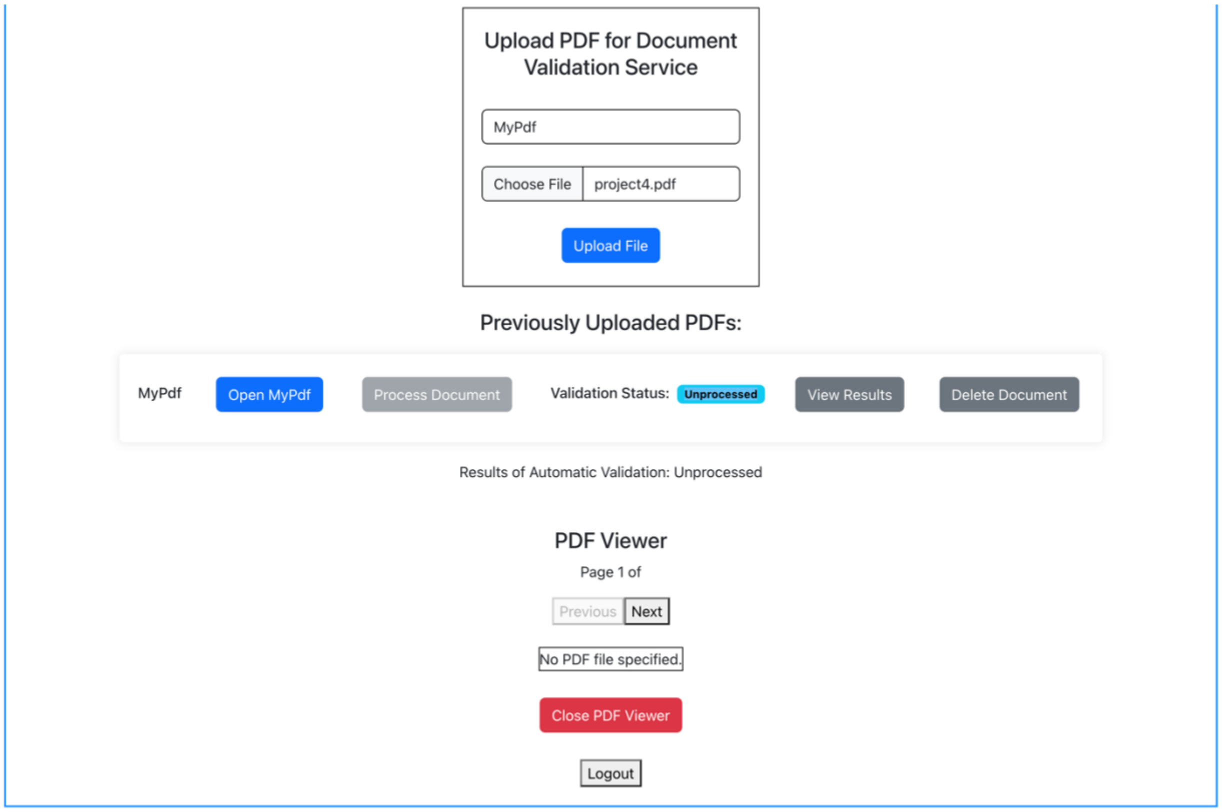
Document Upload

After logging in, the user is taken to the main application page, where they can upload a document. To make the upload process easy for the user, we have containerized the upload in a clearly defined box. In addition, there is an area to view previously uploaded PDF's, and the opportunity for the user to use the PDF viewer to view a previously updated PDF. By including a logout button at the bottom of the screen, we use Nielsen's third-design heuristic of User Control and Freedom. This allows the user to logout at any moment.



Document Verification Options

The following screen gives the user the document validation and viewing options once they have uploaded a document. They can clearly choose to process results, view the document, see their results, and delete the document. In addition, each document has a validation status which tells the user if the document has been processed and whether it is validated or not. When the user clicks show results, text directly below the box which contains the document and its options notifies the user of the results. In this page, we tried to implement Nielsen's heuristic #1, visibility of system status. By showing each document's validation status inline it is easy for the user to determine what is happening with the document and whether it has been processed or not. The option to delete the document allows the user to recover from the error of uploading the wrong document.


Upload PDF for Document Validation Service
MyPdf
Choose File
project4.pdf
Upload File
Previously Uploaded PDFs:
MyPdf
Open MyPdf
Process Document
Validation Status:
Unprocessed
View Results
Delete Document
Results of Automatic Validation: Unprocessed
PDF Viewer
Page 1 of
Previous
Next
No PDF file specified.
Close PDF Viewer
Logout