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:

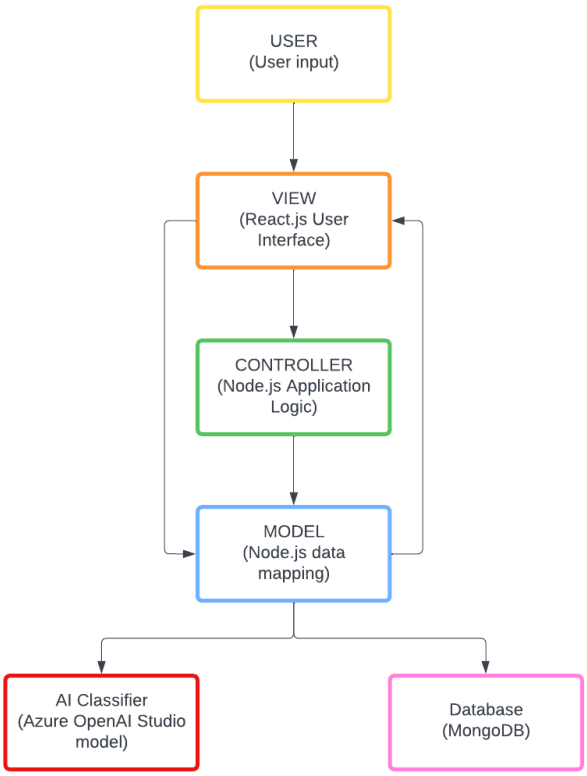

Figure 1. *Static Architecture Diagram (MVC Architecture).*

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:

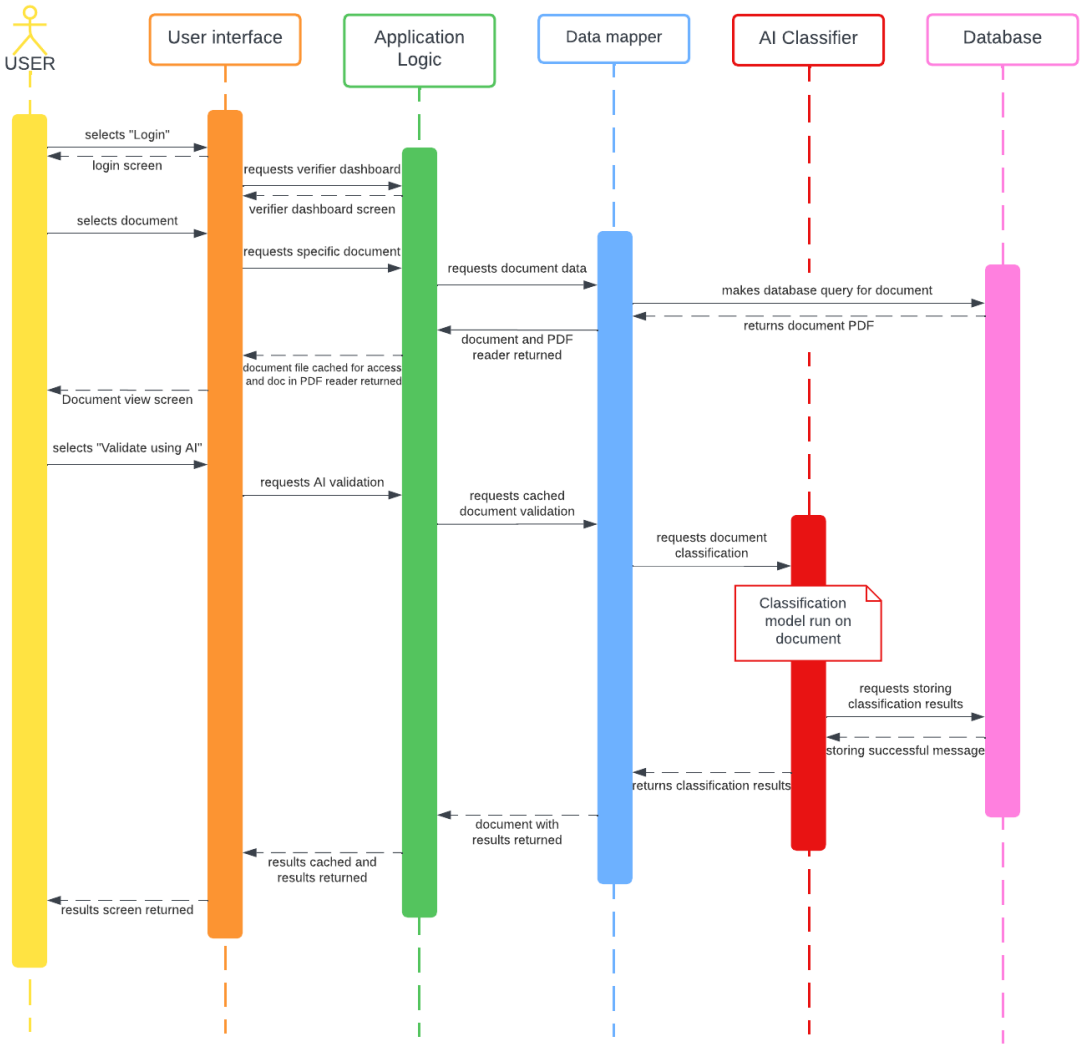

Figure 2. *Dynamic System Architecture displaying a "Running AI Validation" user story.*

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.