**Senior Design Final Report**

Digitization of PD Onboarding & Approval Process

Version 1.0 - 05/14/21

Team Members:

Javier Garcia

Rawad Moussa

Adrian Palomares

Marlito Refuerzo

Christopher Rodriguez

Tabassuma Torosa

Paul Clef Ube

Audelia Valdovinoz

Pierce Wei

Faculty Advisor:

Dr. Chengyu Sun

**Table of Contents**

1. Introduction 2

1.1. Background 2

1.2. Design Principles 2

1.3. Design Benefits 3

1.4. Achievements 3

2. Related Technologies 4

2.1. Existing Solutions 4

2.2. Reused Products 5

3. System Architecture 5

3.1. Overview 5

3.2. Data Flow 6

3.3. Implementation 7

4. Conclusions 8

4.1. Results 8

4.2. Future 8

5. References 9

**1. Introduction:**

**1.1. Background:**

The Los Angeles County Public Defender’s Office is the finest client-centered criminal defense firm in the nation, providing a beacon for evolutionary and revolutionary changes in the justice system. Their mission is to reduce incarceration and the collateral consequences of contact with the criminal justice system in Los Angeles by 2025. With 32 different office locations throughout the country, their mission comprises a team of 1,200 employees - including more than 700 attorneys, as well as paralegals, investigators, psychiatric social workers, and administrative support staff. In order to fulfill their mission, Public Defender has teamed up with California State University to develop a system that involves digitizing the employee onboarding process.

This process allows new employees and contractors to enter their information on a web form, whereupon the data will be stored in a backend architecture. Once reviewed, the data will be sent to the Adobe Sign REST API. The system will then forward the PDF to the necessary people that will sign the agreements and proceed to save the documents on the platform.

The digitization process is initially developed for Windows users, with the plan to expand on to other platforms such as mobile development that include but are not limited to: Android devices, iOS phones, or other means of portable technology.

**1.2. Design Principles:**

The main deliverable here is the application that allows employees and contractors to easily complete required onboarding documents. They will fill out information on a web form, have it saved on the backend, populate a PDF document with the stored data, sign it, and have the completed document saved. The application front-end is designed using AngularJS while the back-end is using Java. The goal is to make the application usable to new incoming employees or contractors, while also able to be managed by Public Defender’s IT from an administrative point of view. The document signing process is going to be very simple, as it will guide the user based on areas that they are required to fill out or sign, and the document will be sent to all necessary parties needed for signatures as well. The application design is made to be simple so that further maintenance or expansions can be made if desired.

**1.3. Design Benefits:**

By designing an architecture that allows Public Defenders to easily access user information and sending it to recipients, we can ensure this would create a very simple but essential method of enrolling new employees or contractors online. The application comes with an Admin page that basically allows Public Defender administrators to have full control of the system and be able to fully oversee any service requests that come through. It will provide a simple interface that can be easily used by developers or handled by Public Defender’s IT members as well.

The application on the other hand, is designed with simplicity in mind. It is designed so that it will be easy for the user to follow and fill out any information that is required, whilst providing a user-friendly interface that keeps the application simple and allowing room for additional improvements if desired.

**1.4. Achievements:**

Over the past academic year, our team has made numerous achievements overcoming limited technological knowledge and obstacles to developing various functionalities. The technical learning curve was a fairly substantial one, but team members managed to overcome many of those initial hurdles.

The front-end subgroup has managed to implement a complete AngularJS system, building multiple web pages that interact with one another to create a cohesive application. The back-end subgroup managed to establish a well-developed Spring Boot architecture, and interface their program with technologies such as SQL, Angular, and Adobe Sign. Additionally, the Adobe Sign portion of the project came with a notable amount of initial research into various aspects of the system (OAuth2 configuration, API interactions, creating document templates, creating Workflows). Beyond these technical achievements, it is of note to mention that all of this was accomplished remotely via the use of collaborative platforms such as Github and Discord.

**2. Related Technologies:**

**2.1. Existing Solutions:**

We have looked into some existing frameworks for creating PDF documents and how they could be applied to our project. These include Apache PDFBox, OpenPDF, and iText. While some of these solutions do allow us to reach our goal with this project, they come with certain restrictions and limitations.

Apache PDFBox allowed us to create new documents, populate documents using the preset field names, add and remove pages. However due to multiple fields being repeated, the difficulty with recognizing the changed field names, and the integration of Adobe Sign; Apache PDFBox would not allow us to seamlessly populate or update the recorded data.

OpenPDF allows us to create new documents, fill fields, and integrate images. However, it was much less supported with limited API instructions resulting in difficulties with achieving a functioning filled form.

iText may have been the most supported framework option with a variety of features. However, we decided against using it initially because we had already created a functioning application with Apache PDFBox and because of the licensing cost.

The main approach of these frameworks was to read the preset field names in order to then populate each field or allow us to enter text by specifying the specific position relative to the document. After the field names have been read we were able to manually enter the field values and fill the provided PDF documents.

This approach worked well for filling and generating a single document. However merging and filling multiple documents became difficult and less efficient when we had several documents to incorporate at once. Additionally the documents were stored on the local device and still needed to be sent for signatures.

Since our application is meant to reduce the time and effort needed for contractors or employees to submit the required documents, we need to use a different approach. Instead we are using features within Adobe Sign, initially intended for the signing portion we were able to create custom templates for each user and assign our own field names to then be filled through the Adobe REST API.

**2.2. Reused Products:**

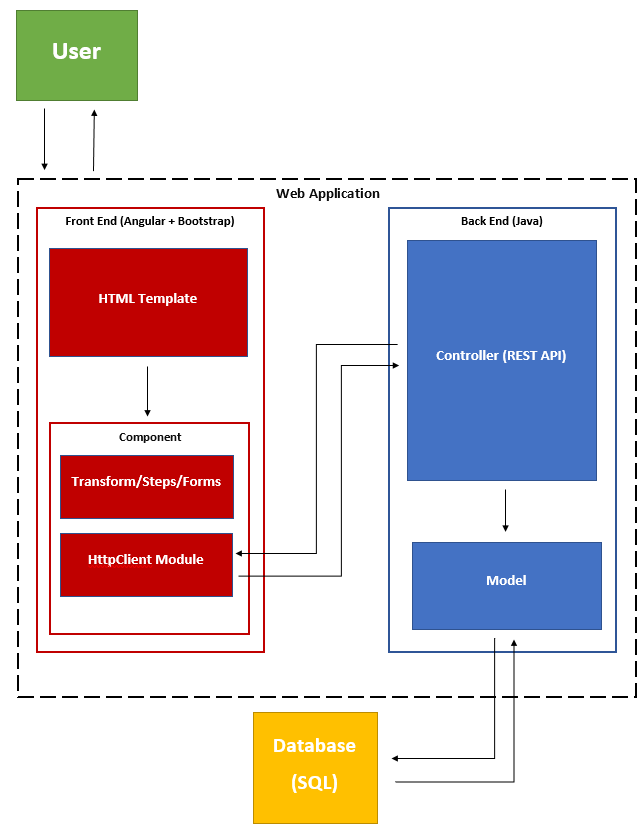
The front end was built using AngularJS while the back end was built using Java and used the Adobe Sign REST API. AngularJS was used to create the UI Elements which will record user input and send it to our SQL database. The SQL database is used to store individual records and then called to fill and generate PDF forms using the Adobe templates.

**3. System architecture:**

**3.1. Overview:**

The system architecture of our project can be typically broken down between two main parts: the Front End (Angular and Bootstrap) and the Back End (Java). With that being said, these two parts are all allocated within the Web Application we have decided to create for this project.

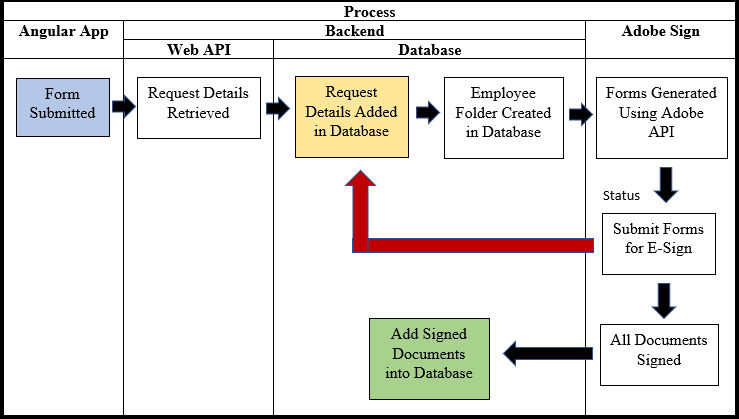
Here is a diagram displaying and elaborating on certain parts of the system architecture:



* **The User:** The individual that is able to make requests and receive responses throughout the Web Application.
* **The Web Application:** Contains a Front End and Back End; this is able to display a simple user interface to the user to interact with and fill out requested forms.
* **Front End:** Contains an HTML template that interacts with Angular components. The HttpClient Module then sends service requests and receives responses from the Back End controller.
* **Back End:** Receives service requests from the Front End, models the data in Java, and then stores the data into the relational database.
* **Database:** Keeps user data and inputted information in storage. It also sends requested data to the model of the Back End.

**3.2. Data Flow:**

Presented in the upcoming picture is a short overview regarding how the program works as a whole as well as detailed information regarding each step.



1. Through the Angular App, users will be able to go to a Web Application that allows them to submit employee and contractor requests. This includes and is not limited to the user receiving a Request Status page (used for the purpose of allowing the user to search certain fields) and a RequestID.
2. Through the Web API, users will have to input information that would be required in the form that they have chosen. With this in mind, the Contractor and Employee forms do contain similar fields but have slight differentiations between the both.
3. Back-End portion of the application adds the request details into the database and creates an Employee Folder for the information. This allows the user to be able to access the document that they have filled out after the signing process is complete.
4. Corresponding forms are generated within Adobe API and are filled with the request details that were previously inputted by the user. The user is then given the task to have their electronic signature added to specified fields. After all fields are correct and have been filled out, the user will then be able to submit the form and have other users that are needed to electronically sign the document. Electronic signatures and other necessary inputs of all users are then stored within the Database and into the Employee Folder as an update to previously recorded information.
5. After all fields have been filled out with the necessary information and electronic signatures, the document is sent back and saved into the database.

**3.3. Implementation:**

The project has been split into two major sections in order to allow efficiency in terms of development: user interface (Web Application) and data management (Java). Respectively, these sections can be defined as the front end and back end of the project.

3.3.1 User Interface

The user interface was made with usability and simplicity in mind. Rather than creating a Web Application that has a bunch of complex functionalities, we made it as a more simple and user friendly interface for users that will be using the UI for the first time.

3.3.2 Data Management

Data and information inputted by the user can only be saved within the database through the use of a working internet connection. Information stored in the database can be later pulled by the user and displayed appropriately.

**4. Conclusions:**

**4.1. Results:**

We have created an application that allows users to fill out information on a user-friendly site hosted by a remote server. The appearance of the application will be very simple which allows the users to know exactly what to enter. It will also come with a save feature in case users would like to continue enrollment at a future time. In the event that the user wants to save, the application provides the user with a unique password dedicated to their page that they can use to re-enter and continue their progress.

We have created a back-end that works with the Adobe Sign API to begin the signing process for users. It will generate a signable PDF that can allow the user to perform their signatures and send it to the necessary parties for approval.. The PDF will then later be saved directly on the Adobe Sign platform, but this may change based on the needs of Public Defender. There will also be an administrative page for assigned users to manage PDFs and perform tasks on the webpage.

**4.2. Future:**

Since the project is developed in a very simple manner, plans for upgrades and expansions can be easily made. Our goal was to provide a simple user interface for the front-end while connecting it to the Adobe Sign API for users to sign forms then further saving it in a database. There can be many potential upgrades in the future for this software and a few that we have discussed that can be done but are not limited to:

* Allowing the application to be accessible on mobile devices such as iOS and Android. This helps give a better user experience and provides better efficiency in the long run.
* Designing a system where administrators can recall previously signed PDFs and look at the information more efficiently rather than searching through the entire database. Being able to withdraw information at a fast, efficient pace encourages better workflow and environment.

**5. References:**

AngularJS Tutorial: <https://www.w3schools.com/angular/>

Law Offices of Los Angeles County Public Defender Website: <https://pubdef.lacounty.gov/>

Adobe Sign REST API Documentation <https://www.adobe.io/apis/documentcloud/sign/docs.html>

Spring Boot REST API Tutorial <https://spring.io/guides/tutorials/rest/>