**Software Design**

**Document**

**for**

**Digitization of PD Onboarding & Approval Process**

**Version 1.2 approved**

**Prepared by** Javier Garcia,

Adrian Palomares,

Rawad Moussa,

Marlito Refuerzo Jr,

Christopher Rodriguez,

Tabassuma Torosa,

Paul Ube,

Audelia Valdovinoz,

Pierce Wei

**Office of Public Defender, LA County**

**14 May, 2021**

Table of Contents................................................................................................................... pg 2

Revision History.................................................................................................................... pg 4

1. Introduction................................................................................................................ pg 5
	1. Purpose........................................................................................................... pg 5
	2. Document Conventions……………………................................................. pg 5
	3. Intended Audience and Reading Suggestions................................................ pg 5
	4. System Overview........................................................................................... pg 5
2. Design Considerations............................................................................................... pg 6
	1. Assumptions and dependencies..................................................................... pg 6
	2. General Constraints........................................................................................ pg 6
	3. Goals and Guidelines..................................................................................... pg 6
	4. Development Methods................................................................................... pg 6
3. Architectural Strategies.............................................................................................. pg 7
4. System Architecture................................................................................................... pg 8
5. Policies and Tactics.................................................................................................... pg 9
	1. Specific Products Used.................................................................................. pg 9
	2. Requirements traceability.............................................................................. pg 9
	3. Testing the software....................................................................................... pg 9
	4. Maintaining the software............................................................................... pg 9
	5. Interfaces for end-users…............................................................................. pg 9
6. Detailed System Design............................................................................................. pg 10

6.1 AngularJS Single Page Application............................................................... pg 10

6.1.1 Responsibilities...................................................................................... pg 10

6.1.2 Constraints............................................................................................. pg 10

6.1.3 Composition........................................................................................... pg 10

6.1.4 Uses/Interactions.................................................................................... pg 10

6.1.5 Resources............................................................................................... pg 10

6.1.6 Interface/Exports.................................................................................... pg 10

6.2 Spring Boot Webapp…………….................................................................. pg 11

6.2.1 Responsibilities...................................................................................... pg 11

6.2.2 Constraints............................................................................................. pg 11

6.2.3 Composition........................................................................................... pg 11

6.2.4 Uses/Interactions.................................................................................... pg 11

6.2.5 Resources............................................................................................... pg 11

6.2.6 Interface/Exports.................................................................................... pg 11

1. Detailed Lower level Component Design.................................................................. pg 12

7.1 SQL Database……......................................................................................... pg 12

7.1.1 Classification.......................................................................................... pg 12

7.1.2 Processing Narrative (PSPEC)............................................................... pg 12

7.1.3 Interface Description.............................................................................. pg 12

7.1.4 Processing Detail.................................................................................... pg 12

7.1.4.1 Design Class Hierarchy................................................................... pg 12

7.1.4.2 Restrictions/Limitations.................................................................. pg 12

7.1.4.3 Performance Issues........................................................................ pg 12

7.1.4.4 Design Constraints......................................................................... pg 12

7.1.4.5 Processing Detail For Each Operation........................................... pg 12

1. User Interface............................................................................................................. pg 13
	1. Overview of User Interface.......................................................................... pg 13
	2. Screen Frameworks or Images..................................................................... pg 13
	3. User Interface Flow Model........................................................................... pg 15
2. Database Design......................................................................................................... pg 16
3. Requirements Validation and Verification................................................................ pg 19
4. Glossary..................................................................................................................... pg 20
5. References................................................................................................................. pg 21

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Date | Reason For Changes | Version |
|  Documentation Team |  12/1/2020 |  Initial creation, small details added. |  1.0 |
| Pierce Wei | 4/19/2021 | Details added | 1.1 |
|  Full Team |  5/10/2021 |  More details added; diagrams as well |  1.2 |

**1. Introduction**

**1.1 Purpose**

The purpose of this document is intended to give a better understanding of the different methods used to collect and store client information in efforts to improve the time it takes to fill and submit required onboarding documents. It will also go over how the system communicates with the Adobe Sign REST API to generate documents and forward it to signers as well as saving it in Adobe Sign.

**1.2 Document Conventions**

N/A

**1.3 Intended Audience and Reading Suggestions**

This document is intended for Law Offices of Los Angeles County Public Defenders IT and developers who may need to maintain the software or make changes to the existing features into the future.The System Overview provided will give a brief description on how the system functions at a surface level.

**1.4 System Overview**

The system requires three functional software components to interact with each other. The front-end consists primarily of a user-friendly UI which allows new employees, vendors, and contractors to fill their required respective PDF documents without entering any repeated field more than once, save their work if need be, and submit the document for the signing process. The back-end maps the corresponding fields to the appropriate PDF form field values to generate the respective forms for approval and signature. The onboarding documents are then generated via Adobe Sign and will be sent to the respective parties that are required to sign. After the final signature is filled, the form is then stored into the database.

**2. Design Considerations**

**2.1 Assumptions and Dependencies**

* Server is required to host web-based application.
* SQL Database is required to store generated documents.
* Back-end Application will rely on Adobe Sign REST API to generate and fill out PDFs.

**2.2 General Constraints**

* Designed for computer use only; operating systems such as Windows should be viable.
* Requires the use of a server in order to maintain sustainability of application.
* UUID is used as a security feature - We assume clients will not have a username/password as new members.
* No sample employees/contractors have been provided - this will affect how we map clients as we assume which fields will be sought after.

**2.3 Goals and Guidelines**

* The end product should speed up form filling processes
* The product should be accessible on any web device
* The UI is user-friendly and easy to follow
* Product should work, look, or “feel” like an existing product that any professional workplace could be able to adopt

**2.4 Development Methods**

This project had not consciously adopted a single developmental method in our approach to software development, but often resembled a typical Agile Development framework. On a weekly basis, subgroups of our team were tasked with meeting certain discrete requirements in our software. We were expected to complete some functional component of the software in a limited amount of time (typically 1 week). Work was completed remotely and collaboratively through the use of Github and Discord.

An Adobe Sign Development method was incorporated into the software to fetch fields located inside our database. With this API, users will be able to receive a OAuth Access Token that allows them to access specific fields they have entered in our web form application.

**3. Architectural Strategies**

* The programming language used for this project includes Java, which is used for the backend development, as well as HTML and CSS, which is used for the frontend development. The main reason for using these languages is because it was requested by the liaisons and are the languages that a majority of the team is used to using.
* For the web form, the team had decided on developing a UI that would be easy to traverse and is user-friendly, keeping in mind users that may not be used to using websites. Rather than creating a UI that has numerous amounts of cluttered information, we decided to use very straightforward buttons that will allow the user to go to where they want to go (see Section 9.2 for screenshots).
* The web form also has an Admin page that allows the IT team to access in-progress forms and view data on them.
* The backend program had to have CRUD capabilities and a REST architecture, so the team decided to settle on Spring Boot as their backend framework of choice.
* An external database is necessary to store information about individual employee/contractor service requests. After some discussion we decided to settle on SQL for the database technology used, as it is easy to integrate with the Hibernate capabilities of the Spring Boot backend. It is connected to the backend program in the configuration files.
* The Adobe Sign API was chosen for the signing process, as it streamlines the process from a user perspective. When prompted to sign, users will type their e-signature once, and simply click on the remaining fields that are necessary to be signed across the document.
* Possible future plans for extending or enhancing the software include and are not limited to making the UI more accessible to other users and forms and adding more functionalities and accessibilities for the user. However, by adding these capabilities, the UI may become a bit more cluttered and slightly more difficult to traverse.

**4. System Architecture**

The flow diagram of the system architecture is as follows:



* New Employees or Contractors interact directly with the Angular SPA (Single Page Application) on the front-end.
* Administrator users interact with the same front-end application after password authentication.
* The Angular SPA sends user data in JSON format to the Spring Boot Webapp on the back-end, which receives the data via the REST API.
* The Spring Boot Webapp employs a SQL database where all the data is stored and managed. When a request is marked submitted, the Administrator user is notified of the incoming request in need of review.
* The Spring Boot Webapp must be configured to connect to an Adobe Sign account via OAuth2 security.
* Once a request is reviewed and assigned approvers by the administrator, the data is populated into an Adobe Sign template via the Adobe Sign Web API.
* When it is their turn to sign, the requestor and approvers are notified via email. They may then visit the Adobe Sign platform to sign the Agreement document.
* Finished documents are stored on Adobe Sign.

**5. Policies and Tactics**

**5.1 Choice of which specific products used**

To develop the Angular Single Page Application, Visual Studio Code was utilized. User information submitted in the form is stored in a SQL database. Spring Boot application in Eclipse EE was developed in the back-end to provide a service that connects the front-end and the SQL database. Adobe Sign was our product of choice for the approval workflow. These technologies were decided on early in our project development, so there was not much negotiation on the possible alternatives.

**5.2 Plans for ensuring requirements traceability**

Our primary plan to trace the progress of various requirements involved constantly monitoring the “Project Requirements” page hosted on CSNS, and comparing it to the existing version of our program.

**5.3 Plans for testing the software**

Adding test data into the web form, submitting the form so that it shows up in the back-end, sending it to Adobe Sign to initiate the signing process that should forward the signable document to necessary users, then saving the document within Adobe Sign.

**5.4 Plans for maintaining the software**

Plans to maintain the software will be left in the hands of the Public Defender IT Team. They may include contingencies for long-term OAuth2 implementation, online hosting of applications, testing in production environments, and adopting an Adobe account with more robust permissions.

**5.5 Interfaces for end-users**

Employees, contractors, and administrators work with the Angular SPA interface. Requestors and approvers interface directly with the Adobe Sign platform.

**6. Detailed System Design**

**6.1 Angular Single Page Application**

**6.1.1 Responsibilities**

Angular is primarily responsible for providing users with a clean interface and smooth process to enter their information. It’s also responsible for formatting the user’s data with the correct field structure in JSON form to send to the Spring Boot back-end program. It will then generate a signable PDF that will create an efficient method of registering new users into the system. This provides a smoother transition for incorporating new clients into Public Defenders and can be easily managed by IT.

**6.1.2 Constraints**

It is assumed that only users can access unsubmitted information based on their unique request number. The request number must be remembered since there is no request number retrieval when forgotten or lost. It is also assumed that the signer’s email has been entered correctly, so the Adobe Sign process proceeds smoothly. Not only that, the Adobe Sign process can only be accessed by admins.

**6.1.3 Composition**

Angular contains three subcomponents: HTTP service, Angular router, and the form (user and admin). The HTTP service facilitates communication with the Spring Boot back-end web app. The Angular router enables multiple views and allows navigation between these views. Lastly, the form enables the user to perform data-entry tasks and admins to login and update the user’s information and profile.

**6.1.4 Uses/Interactions**

The form and HTTP service is used when the user clicks on “save” or “submit,” or when admin updates any information, which then sends data to the back-end. The form and http service is also used when admin logins with their username and password. The router takes into effect when the user goes to another page by clicking “next,” “home,” or any other navigation button that changes the view.

**6.1.5 Resources**

Being a Single Page Application (SPA), the Angular application does not depend on many resources. The only resource the application will depend on is the Spring Boot server that was developed alongside the application. The server is where the data is retrieved from as well as where the data is sent to. The application will still run without the server running, but not much functionality will be available other than filling out the form. Furthermore, the admin part of this application will not be accessible since the login part depends on the server for verifying the login password.

**6.1.6 Interface/Exports**

The “app” container contains components of different form views (ex: admin, contractor form, employee form, etc) and controllers per their component that controls the navigation, user’s data-entry and mouse interaction, and communication with the back-end.

**6.2 Spring Boot Webapp**

**6.2.1 Responsibilities**

The Spring Boot webapp is responsible for collecting user onboarding data in JSON form via the AngularJS front-end program. It interacts with an SQL database via Hibernate to store this data in relational form. Additionally, the program is responsible for sending various emails to a number of parties involved. Finally, the program also interfaces with the Adobe Sign API to populate template documents hosted on the platform and denote approvers that would have to sign the completed document.

**6.2.2 Constraints**

It is assumed that the backend program interfaces directly with the Adobe Sign API every 60 days - otherwise the Refresh Token will go out of date. It is also assumed that the SQL database used is capable of storing tuples with many fields included. The backend program operates on the assumption that the data received from the front-end program will correspond to the field structure on the back-end program.

**6.2.3 Composition**

These REST endpoints are mapped to the appropriate SQL relations: Service Request, Admin, Div Chief / Manager, Dept. Info Security Officer, Application Coordinator. The following SQL relations are for internal use only: Field, Form, Mapping.

**6.2.4 Uses/Interactions**

This application interacts with the AngularJS front-end (see section 6.1) via HTTP requests in JSON form. It also interacts with an SQL database via the Hibernate framework. Additionally, it interacts with the Adobe Sign API via HTTP requests in JSON form.

**6.2.5 Resources**

An SQL database is used to display and store information from filling out user information to generating a PDF. The database will be moved to AWS by the IT department in Public Defenders in the future.

**6.2.6 Interface/Exports**

The “model” package contains class representations of various objects (ex: ServiceRequest, Admin, Mappings, etc.). The “controller” package contains logic for the public endpoints of the back-end application. Much of the program logic lies here. The “service” package contains a file with logic for sending emails from the application. The “signapi” package contains Adobe Sign credentials and logic to interact with the Adobe Sign API. All HTTP calls to Adobe Sign’s REST API are handled here.

**7. Detailed Lower level Component Design**

**7.1 SQL Database**

**7.1.1 Classification**

This Component may be classified as a *database*.

**7.1.2 Processing Narrative (PSPEC)**

N/A

**7.1.3 Interface Description**

Developers may interface with the SQL database via a terminal program (such as MySQL) or a helper GUI program (such as Adminer).

**7.1.4 Processing Detail**

**7.1.4.1 Design Class Hierarchy**

N/A

**7.1.4.2 Restrictions/Limitations**

The chosen database must be compatible with the .sql configuration file we have created. This involves support for relations with many tuples and fields.

**7.1.4.3 Performance Issues**

On submission, individual data entries do not present many performance issues. If there is a higher volume of data being entered into the database, some slowdowns may arise. Scalability should be addressed in future iterations.

**7.1.4.4 Design Constraints**

 N/A

**7.1.4.5 Processing Detail For Each Operation**

The backend program creates and manipulates data on the SQL database via the Hibernate framework. Developers specify operations in Java, and Hibernate translates these operations into discrete commands in SQL when interfacing with the database.

**8. User Interface**

**8.1 Overview of User Interface**

The UI will be very simple and should easily guide the user precisely to what field they are required to enter. The initial first page should prompt the user to enter a password or start a new form. Upon starting the new form, there will be fields that indicate what information the system needs for the user to enter. The user also is given the ability to temporarily save the form whilst given a unique password that they can enter when they revisit the page if they wished to continue where they left off. The form will also request whether the user is an employee or contractor which is pretty straightforward and once the information has been filled out, the user can send it in for the signing process.

**8.2 Screen Frameworks or Images**

1. User Initial Login Page
2. Contractor or Employee Page
3. Start a New Form or Continue a Form
4. User Enters Information

**8.3 User Interface Flow Model**

The following is the flow model for the front-end user interface:



**9. Database Design**

Our back-end Spring Boot program makes use of an SQL database. Multiple REST endpoints are associated with various tables in the database. The relations are as follows:

Public (REST Endpoints): Service Request, Admin, Div Chief / Manager, Dept. Info Security Officer, Application Coordinator

Private (Internal): Mappings, Forms, Fields

**Public Endpoints**

1. Service Requests

|  |  |  |
| --- | --- | --- |
| **METHOD** | **ENDPOINT** | **DESCRIPTION** |
| GET | service\_requests | Gets all the service requests in the database |
| GET | service\_requests/{requestNumber} | Get a specific service request based on requestNumber |
| POST | service\_requests | Add a service request |
| PATCH | service\_requests/{requestNumber} | Update a service request |
| DELETE | service\_requests/{requestNumber} | Delete a service request |

2. Request Statuses

|  |  |  |
| --- | --- | --- |
| **METHOD** | **ENDPOINT** | **DESCRIPTION** |
| GET | request\_statuses/{id} | Get the service request's request status |

3. Admin-related Endpoints

|  |  |  |
| --- | --- | --- |
| **METHOD** | **ENDPOINT** | **DESCRIPTION** |
| GET | admin/service\_requests | Gets all the service requests (condensed to 4 fields) in the database |
| GET | admin/service\_requests/{requestNumber} | Get a specific service request based on requestNumber |
| PATCH | admin/service\_requests/{requestNumber} | Make completely unrestricted updates to any service request fields |
| PATCH | admin/reset\_password | Update password with new-password header |
| GET | div\_chief\_managers | Get all Div Chief Managers |
| GET | div\_chief\_managers/{id} | Get a Div Chief Manager based on id |
| POST | div\_chief\_managers | Add a new Div Chief Manager |
| PATCH | div\_chief\_managers/{id} | Update a Div Chief Manager |
| DELETE | div\_chief\_managers/{id} | Delete a Div Chief Manager |
| GET | dept\_info\_security\_officers | Get all Dept Info Security Officers |
| GET | dept\_info\_security\_officers/{id} | Get a Dept Info Security Officer based on id |
| POST | dept\_info\_security\_officers | Add a new Dept Info Security Officer |
| PATCH | dept\_info\_security\_officers/{id} | Update a Dept Info Security Officer |
| DELETE | dept\_info\_security\_officers/{id} | Delete a Dept Info Security Officer |
| GET | department\_heads | Get all Department Heads |
| GET | department\_heads/{id} | Get a Department Head based on id |
| POST | department\_heads | Add a new Department Head |
| PATCH | department\_heads/{id} | Update a Department Head |
| DELETE | department\_heads/{id} | Delete a Department Head |
| GET | application\_coordinators | Get all Application Coordinators |
| GET | application\_coordinators/{id} | Get an Application Coordinator based on id |
| POST | application\_coordinators | Add an new Application Coordinator |
| PATCH | application\_coordinators/{id} | Update an Application Coordinator |
| DELETE | application\_coordinators/{id} | Delete an Application Coordinator |

**10. Requirements Validation and Verification**

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Satisfied By** | **Testing Method** |
| Gather relevant employee/contractor onboarding data. | Angular web form (front-end). | Compare PDF fields to web form fields. |
| Store user data from the web form. | Backend SQL Database. | Run both programs, complete sample form, check SQL database. |
| Allow PD members to review individual service requests. | Admin authentication, notifications, and request review page (Angular front-end). | Login to “Admin” page of Angular front-end, access the “Review Request” page of any particular request. |
| Allow approvers to be selected for a particular request. | “Add approvers” page for Admin users. | Add a sample approver, select the approver to use for a sample request, and see if they are notified at their turn in the signing process. |
| Implement a signing process for the requestor and various approvers. | Integration of Adobe Sign API with the back-end program. | Simply submit a request to Adobe Sign as an Admin user and ensure that all parties involved receive Adobe Sign notifications. |

**11. Glossary**

Adobe API - Adobe Application Programming Interface

AngularJS - JavaScript-based open-source front-end web framework used in developing single-page applications

ISD - Internal Services Department

IT - Information Technology

PDF - Portable Document Format

REST API - Representational state transfer is a software architectural style which uses a subset of HTTP. It is commonly used to create interactive applications that use Web services. A Web service that follows these guidelines is called RESTful

SRS - Software Requirements Specification

UI - User Interface

UUID - Universally Unique Identifier

**12. References**

Software Requirements Specification for Digitizing the Employee/Contractor Onboarding Process Version 1.3 approved

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>

Adobe Sign Sample REST API <https://github.com/adobe-sign/rest-api-sample>

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