Upgrade of Tenant and Owner Portals to Housing Authority Website

Prepared by Juan Rojas
Simon Bach
Michael Schleicher
Juan Orozco
Ravin Bhakta

Advisors: Russ Abbott, Jungsoo (Soo) Lim

Los Angeles County, Housing Authority

May 10, 2019
1. Introduction

The sponsor for this project was the Housing Authority of the County of Los Angeles. The project consists of upgrading the Tenant and Owner portals, specifically for section 8 families. The portal is a way for users to fill out and submit online forms to maintain eligibility for the housing programs. The current website was built in 2010, so it does not meet today’s technology standards and needs an upgraded modern, user-friendly user interface. The Los Angeles County uses C# with ASP.NET for their web services; therefore we are limited to making a system that must also comply with ASP.NET. The current system also lacks a SQL relational database where information is stored in different tables. All information is currently stored in one table in XML format, meaning SQL queries cannot be performed.

One of the significant challenges that had to be tackled early was the database design. The original database stored all ten types of form submissions in one table, one column, in XML format. The problem with this is that we could not perform SQL queries on the data. For example in a question where the user can check many checkboxes pertaining to their answer, the system stored all checked boxes as a single string only separating the choice with a pipe as a delimiter. A solution to clean up the database was to design a database with an entity-relationship model. We created tables for each form, and we created scripts to extract the XML data and input it to its respective table.

The motivation of our proposed solution stemmed from the idea of decoupling the front end from the back end while still adhering to the rules of ASP.NET. As a result, we created WEB APIs to handle the storing and retrieving of information to the database. These WEB APIs are a set of functions that contain information on how to store data on to the database. For the
backend, we used ASP.NET to create WEB APIs and serve the web pages. An advantage to using WEB APIs is that they can be reused in a phone application. For example, in the future, if the Los Angeles County decides to make a native mobile app, they can use the existing APIs and incorporate them into the app to perform the same basic functions. On the frontend, we used Bootstrap for the design and styling of the webpage. One of the main features that are missing is the ability to adapt to mobile devices. It is essential, in this day and age, for websites to be able to adapt to mobile devices because it makes it much easier for the user to access their account information anytime, anywhere. The renovated portal uses Bootstrap 4.0, which is an open source CSS framework that allowed us to make the portal look modern and to resize the page onto a mobile device. We used Vue js, which is a JavaScript framework that works with our website, by providing dynamic information. Vue offers accessible ways to display the information onto the page.
2a. System Architecture

The three components of the system architecture are the database, Web API with ASP.NET, and Vue js. The database stores all tenants’ and owners’ personal information. The Web API with ASP.Net is the backend controller which handles passing around data from the database to the user. The Web API handles all action required on the portal, which means that it also handles create, read, update, and delete needed on the database. The frontend framework used HTML with Bootstrap 4.0 for styling, and Vue js that helps render information using virtual DOM (Document Object Model). The use of a virtual DOM helps by changing the data model and updating the information dynamically without the need of refreshing the page.
2b. Database Design

One of the main objectives of this project was to redesign the database that stores all the user’s submitted forms. The current portal just used a single table for all the submitted forms and stored all the forms in an XML encoded string. This allowed the backend to very quickly store every submitted form, for the XML encoding that was stored was just simply the front end’s XML encoding of the user’s form data that was created to send the data to the backend. This also allowed the backend to retrieve the whole XML string without having to do any processing, and send it to a front end to be displayed. The downside was that to do any queries on the form data submitted would require a lot of string processing, which will not scale reasonably as more completed forms were submitted.

The current database was like storing all the submitted forms into a single filing cabinet. The plan to redesign the database was pretty much to organize each form into their own filing cabinet and then doing some processing to reduce redundancy in the data being stored for each form. The most common reduction we found was in the representation of checkboxes. For example, in the Request Voucher Extension form the following checkbox is presented to the user:
We converted the checkbox into its own table, resulting in key-value pairs for each object in the checkbox. This allowed us to store a primary key integer representing each option the user selected. This design allowed for queries on the user submitted form data, but it required a lot of work to convert all of the old form data currently in the database.

We created SQL scripts to convert the form data already stored in the old database. First we had to parse the XML to extract every form entry. Then we noticed the many to many relations, like the checkbox above, were stored by concatenating the string displayed for each selected option with a ‘|’ character as the delimiter. So we had to split those options on the ‘|’ character to extract each option, then converted them to the corresponding primary keys. This resulted in generally three scripts for each of the ten different forms: one to create a table of all the options in every checkbox or drop down on the form, one to convert the rest of the form, and then the last one to correlate each form with every option select in the many to many relationship.
3. Results and Conclusion

Our initial objectives were to redesign the user interface. The two main aspects we focused on was to make a modern user interface and to add the ability for the web page to adapt to mobile devices. To accomplish this we mostly relied on Bootstrap, which helped us with the look of the website. As for the database, we successfully migrated the XML encoded data into a separate database with an entity-relationship model. Having a table for each form allows for the database to be more easily readable. Some steps the Housing Authority can take next, for the project is to create an actual phone application. The Web APIs are already built and can be reused in the mobile application. If they choose to use the current Web APIs then the security of the APIs can also be improved. This will help for future developments of the web application into applications for phones and tablets.