Pay It Forward

Final Report

MAY1706
City of Ames
Swamy Ponpandi

Evan Blackwell - Team Lead
Christian Klein - Key Concept Holder
Carter Thayer - Architect
Grace Winchip - Communication Lead
Brody Concannon - Webmaster

may1706@iastate.edu
http://may1706.sd.ece.iastate.edu/

Revised: April 23, 2017

Contents

Introduction

Project Statement

The Pay it Forward project will result in a web-based platform hosted by the City of Ames to connect used-item donors with appropriate donation outlets. Outlets will be able to create accounts and make changes to their publicly available information as approved by administrators of the system. After that, outlets will be able to select which items they accept as donations. Citizens who wish to donate items can input the things they have available then search for the best outlet for them. Additional features are included to incentivize the use of our website. This application should be easy and fast to use from both web and mobile browsers.

Purpose

The City Council has made it a priority to reduce waste going to landfills, and they made our client responsible for seeing to this challenge. In order to achieve this goal, our client wants to encourage the community to reuse and recycle as often as possible. Our application will provide the residents of Ames tool to simplify the process of finding an outlet for donating their items for reuse. By making this step easier, it is expected to increase the amount of items that are reused within the community; thereby, decreasing the amount of waste being contributed to landfills. If this project achieves its goal, the non-profit organizations of Ames will see an increase in donations to help them succeed in fulfilling their missions while creating a healthier and cleaner system by lowering traffic to landfills.

Project Design and Implementation

Requirements

Non-functional Requirements

- Must be mobile accessible with logical formatting
- Should be easy to use
    - Provide a simple interface for users of all backgrounds
- Must be integratable into the existing City of Ames software platform
- Allows guests access to search function
- Shall be portable for use in other locations

Functional Requirements

- Provide a list of all donation centers
- Provide profile pages for each donation center
    - Should include information about hours, contact information, location, items accepted, if pickup/delivery is available, etc.
- Give users the ability to search for donation centers
    - Allow users to enter item types to donate by category (including other)
    - Allow user to save previous donations to recall at a later time
- Allow users to produce a summary to be saved, printed, or emailed
    - Should include information such as hours, locations, and items accepted for each donation center in search
- Provide a method to create donation centers
- Allow existing donation centers to update their information
- Allow admins to:
    - Send emails from the system to donation center owners
    - Approve and delete donation centers
    - Edit the purpose and other site details
- Use Google Maps integration
    - Gather user location to provide distance to donation centers
    - Donation center profiles have a map marking their location
- The system shall store donation history and search history for the donors access later
- The system shall attach values to donations and compile them into a report for taxation purposes

Platform and Technologies

The City of Ames specified that we must be able to integrate with their existing Windows platform. We used a Microsoft SQL server for our database, and the project was built using ASP.net. In order to simplify the use and design of our database, we utilized the Entity Framework, which provides an easy to use API for reads and writes. In conjunction with Migrations, Entity allows developers to create and update the schema of a database modeled by the public properties of C# classes. The front end was implemented with the use of HTML and Bootstrap.

Design Overview

The website is comprised of multiple pages, each with a specific function, rather than a single page with an updating content section. The multiple-page approach is more familiar among the team and promotes modularity throughout the product. Microsoft's ASP.net allows for a few development strategies: to tightly couple the frontend and backend, or to use a messaging system between the front and backend. We decided to pursue the tightly coupled route, as it is simpler and seemingly more appropriate for a smaller website such as Pay It Forward.

Figure 1 provides a high-level view of the website created by our team after meeting with Merry and analyzing her drafts. Each box in the figure represents a webpage, and each arrow represents navigation from one page to another. Each of these pages are fundamental to the site's functionality. Each of these pages has its own aspx and cs file to detail its appearance and abilities.

Back-End

In Appendix IV, Figure 2 provides a more visual representation of the schema of our database. As previously mentioned, we used the Entity Framework to manage our database needs. It's a tool that easily allows you to read and write to your server and to effortlessly add and drop columns and tables to your database. In order to use this tool, you have to specify a database context and which classes should appear in that database. There exist many different tags you can use to make Entity do what you want. For example, there are options to not include certain public properties as columns and to automatically produce unique ID numbers for new objects.

The core feature of our website is the ability to enter a list of items and in return receive a list of donation centers that will accept the items that you want. All other functionalities are built around this idea in order to provide more value to users. To use this feature, a user does not have to be logged into the website. In order

to make finding items easier for donors and for donation centers, we organized the items into categories. Each item belongs to only one category, but each category has many items. We recently switched our design, so queries are done on the item type instead of on the category. This was done to allow more customization of what items donation centers will accept. The items are stored as a session variable, so the next page can find relevant donation centers based on the donation.

Donation centers are responsible for signing themselves up to the website. In order to signup, the owner must have an account in our system. The User class will store any donation centers associated with a user and allow them to make edits. Upon creation, the donation center will not be visible until the admin approves the request; however, edits on a donation center will be immediately available if the site is already visible. Admins will be notified of all requests to create or edit a donation center and have the option to email the requesting user. If a request is accepted, the center will be viewable to donors. If the request is rejected, the business will be hidden and not show up in search results.

Users store a lot of valuable information relevant to individuals who access our site. If a user is logged into the system when they donate, their donation can be stored for their later use. Donations are stored in the database, but in order to limit memory usage, we used a formatted string to log what items were donated. The item type, quantity, and the center donated to are all stored for the User. This history can be viewed on the User's page, and they can receive a valuation of their donations. We received these values from Goodwill and Salvation Army.

Users with administration privileges have access to more tools on the site. They control the site through requests that get sent when donation centers are added or want to edit their information. Additionally, they can update, add, and delete items and categories to the database through a form.

Front-End

As an ASP.net website, Pay It Forward's front-end design consists of HTML, CSS, and C#. Not everyone in the group had previous experience with these languages, but with the plentiful learning resources, the daunting learning curve was overcome in decent time.

The ASP.net framework slightly adjusts the HTML design, but for the most part it is standard HTML. In any case where the HTML interaction is strictly client-side, the Pay It Forward development is 100 percent equivalent to standard development. In cases where HTML interaction requires any sort of server-side operation, the HTML needs to sway from standard development and include

things such as runat=”server” and maybe even need to be renamed to asp:Button, asp:Panel, etc.

The CSS for our site uses Bootstrap for core features such as the header that is present on all pages. Best practices for creating a Bootstrap header are to copy and paste the standard header, and then rename and adjust the elements. This is not a difficult process, so we followed best practices and used this approach. Pay It Forward does not include any wild or fascinating CSS, so the entirety of page-specific CSS is included at the end of each HTML file rather than within an additional separate file.

The majority of our C# code is included in the back-end, but sometimes pages need to be populated with asp controls by the “.aspx” file. An example of this is the ViewDonationCenters page, which reads the “items” session variable and displays all donation center that accept those items. The user may then click the center that we wish to see more information for. This needed to be done by C# for two reasons: we had to read the session variable (which must be done server-side) and we needed to define a click function that passes in a donation center name in order to correctly redirect on user click. Instead of populating the page with HTML ’s, the page had to be populated by asp elements such as asp:Panel and asp:Table. This design allowed for the desired user behavior.

Standards

Throughout the development of this application, we stuck to some implicit standards for code style and database management. Given the technologies we are using for Pay It Forward, we adopted some those guidelines and practices.

Code style was not strictly enforced, but most of the C# code follows the general styling used in most C# applications. This includes naming schemes, casing, types of methods put on classes. In addition, when working with the front-end, we used ASP.net technologies wherever possible to simplify the usage and interaction between the back and the front.

When working with the database, we used the Entity framework. This allows us to use C# objects to represent our database tables. Using Entity was enforced in our application and no direct SQL calls are used. Additionally, Entity provides a migration tool for database. Each database change should have a migration, so that we can these on the database on get the correct format. This, also, allows for rolling back the database in necessary.

Testing Process and Results

Due to the website nature of Pay It Forward, there were not very many cases where systematic testing was entirely feasible or even especially effective. A few unit tests were created in order to test basic database pushes and pulls, but the majority of our testing was behavioral.

Behavioral testing tools exist, such as JBehave, Cucumber, and Codeception, but we deemed the setup time and learning curve of these tools to not be worth the minor efficiency that they provide for our relatively small application. Additionally, Pay It Forward does not contain extremely intricate or complicated user functionality, so it was simply easier to run the tests ourselves.

Our behavior testing was performed after each feature was added to the website. Since we were performing tests ourselves, we made sure to take note of any possible edge cases before each session of testing. We then ran the application to try and break our own code, noting any peculiar behavior that occurred. If a problem was detected, it would be soon fixed.

To further ensure our application works and meets the high standards set, we continually met with our client throughout the semester. Each time, we brought her new features to look at and play with. Not only did these meetings help us keep on track, we were able to see how a user would use and think about our application.

Appendix I: Operation Manual

Application Deployment

The following process was initially done on a Windows Server 2016 machine through Iowa State ETG. Processes may be different based upon your environment and preferences.

1. Install the appropropriate components on the server.
   For our server we needed to install the following:
   a. Web Server role and management tools. These can be installed via “Add Roles and Features Wizard”. A screenshot of the installed roles is available in Appendix IV, Figure 3.
   b. .NET 3.5 and .NET 4.6 features. These can be installed via the “Add Roles and Features Wizard”.
   c. Microsoft Web Deploy 3.6. This is necessary for deploying the application to IIS. This is available through the Microsoft Web Site or the Microsoft Web Platform Installer.
2. Create a Web Deploy Package using Visual Studio. Deployment is also possible with Web Deploy over the internet. Due to connection limitation/difficulties, we used the web deploy package feature.
   a. In Visual Studio, open the PayItForward solution. Right click on the PayItForward solution and select Publish.
   b. A profile should be included with the project called “CustomProfile”. You may also create your own if you choose.
   c. Select Publish.
   d. Transfer the .zip file to your server.
3. Install the application on your server.
   a. In IIS Manager, find the appropriate site on your left hand side bar.
   b. Right click it, select Deploy, select Import Application.
   c. Browse to the package on your server and hit next.
   d. Select the package contents, we did not need to remove any of the contents.
   e. Hit next and enter settings for the application.
      i. Enter the application path. We used this as our Default Web Site, so we removed the default path from the text box.
      ii. Enter a Connection String for the DatabaseModel. This should be formatted as per the Web.config connection string specification and include database connection information.

f. Select “Yes, delete all extra files…” to replace any application files that may already be on this site.

4. Browse to the application on the web to ensure correct deployment.

Admin Operation

The administrative operations are designed to only be able to be accessed by users with an administrative role. These operations include handling pending requests, viewing request history, managing donation centers, and managing items and categories. Many of these features are crucial to the business process, allowing our web application to function as desired. For example, when a donation center is created a notification is stored in the database; an administrator can then approve or deny the request as they see fit, and the server will update the databases to make the newly created donation center visible to users.

All of these are operations help the administrator have an easy to use interface to manage tasks the application depends on without the need for IT support. The “Admin Action Center” allows the application to support the donation centers and the overall goals.

Appendix II: Alternative Design Versions

Single Page Application

Some recent web development trends opt for building single page applications with a Javascript framework. This application would be backed by a RESTful web API. This design was discussed within our team early on.

The primary advantage to this design is the reusability of the backend application. This API would primarily be used for the front end web application, but could be reused programmatically be users (if we released the API spec) or a mobile application. Additionally, the single page application model would allow use to choose whatever Javascript framework we would like (while keeping the back-end in C# for ease of transition to the City of Ames).

We ruled this design out for a few reasons. Firstly, this would mean two sets of code. Our team would have to split ourselves to designate front-end and back-end developers or everyone would have to learn and be familiar with two technology sets. With the unified approach we took, the technology stack is self contained. One set of code would be worked on throughout the team. In addition, we know that the programmers at the City are fully aware and worked with the full stack of the ASP.net technologies (including Bootstrap on the front-end).

Appendix III: Other Considerations

Learning ASP.net/C#

When the requirements of Pay It Forward, we were presented with the current technology stack that the City of Ames uses. The Microsoft Web stack seemed daunting to work with for most of our team, who did not have experience with it.

Over the course of the year, we were able to dive right into how these applications are built and function. It has been a joyful learning experience. As software developers, we find working with new, unfamiliar technologies interesting. Now, having the experience of being dropped into a different stack will come in handy when called for similar challenges in the future.

Single Development Database

To keep set up time minimal and to dive straight into development of the application, we decided to use a single hosted database that we all worked on. Quickly, we found that this was a naive approach.

During first few weeks, we saw changes to the database that our master branch was not compatible. This was due to team members making changes to the database, without completing and committing the changed code. These issues were relatively minor, but caused some initial confusion. We all adapted and made sure that any database changes had code reflected on master branch of our code repository.

Another issue occur with this approach, when we attempted a “code-a-thon” one afternoon to complete part of the project as a group. Early, during this event, an accidental drop occurred to the entirety of our development database. This caused a bit of set back for our coding event, as two our team members spent most of that afternoon restoring the database to it’s former state. Luckily during this process, we were able to reinforce what the process looks like to get our application running off an initially empty database.

In retrospect, individual local development databases would have been preferred, even with the setup time. In a project that goes longer or has more team, local development databases would be necessary. Then when committed to master, a continuous integration environment could apply any changes to remote development database and running development application.

Appendix IV: Figures

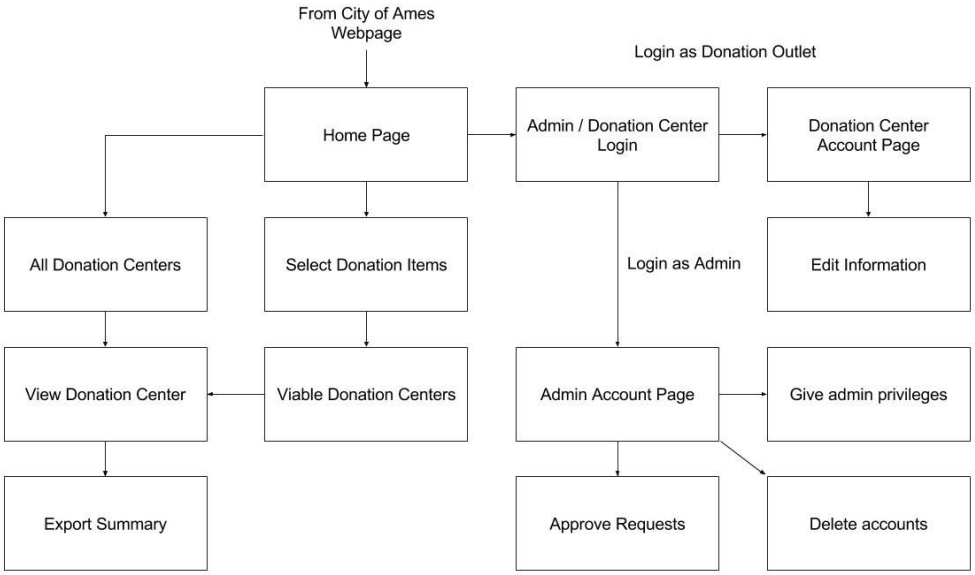

Figure 1

Overview of the website structure.

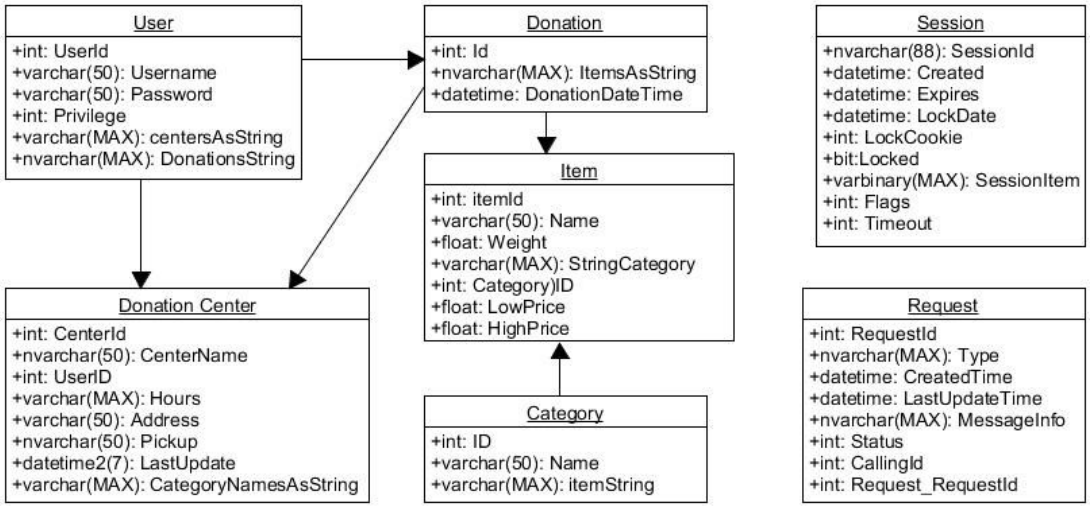

Figure 2

Schema of the tables used in our database. Slight changes may occur with ongoing development of the project.

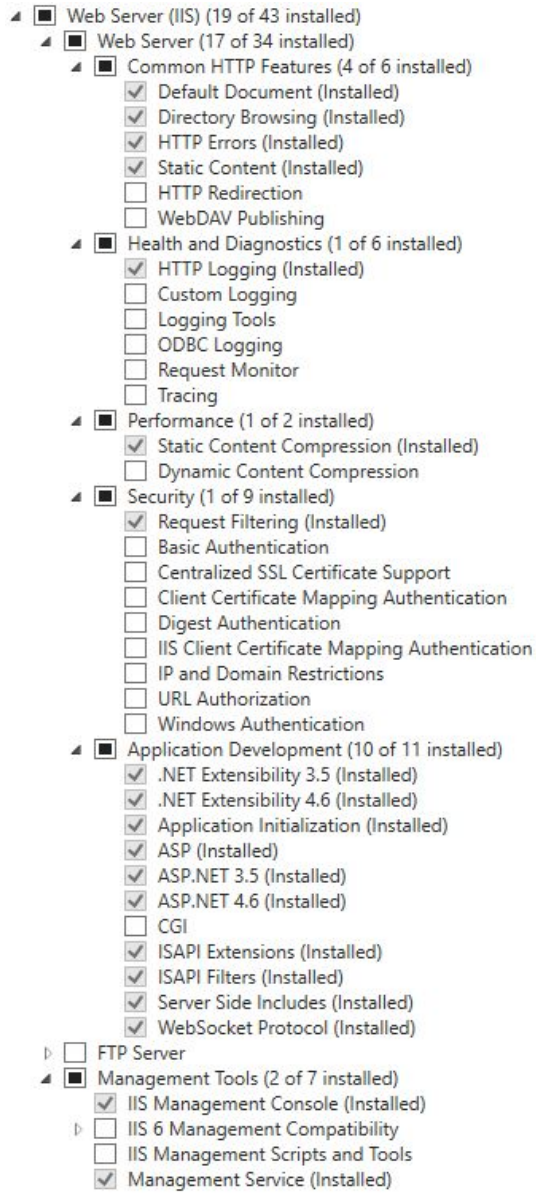

Figure 3

Screenshot of installed Windows Rolls