Dillon Puglisi

CS-499-X2150 Computer Science Capstone 20EW2

12/8/20

Final Project

**Self-Assessment**

Choosing the artifacts I was going to update for this project was a difficult process to begin with, as my time in this program has been short and none of the projects I’ve completed within it could be considered noteworthy. I wrote much code at my previous school before I transferred, but much of it was lost in the transfer. Eventually, I realized that my best option wasn’t a school project, but rather the personal project I’ve been developing throughout my entire career as a programmer. As soon as I learned to build websites-- starting with html, javascript, php and sql interactions-- I started building myself a web-based tool. This tool would eventually help me better organize and manage many aspects of my life, consolidating the features of several pre-existing digital and paper-based tools. As my skills grew, and I learned new languages and tricks, both from professional and academic work, I incorporated them into my website. It is a working manifestation of my programming career and growth, and therefore the perfect choice for this capstone.

A good programmer learns wherever and whenever they can. I took lessons from my school- and professional work, and from the mistakes I made, and strengthened my site every chance I had. When I learned how to implement a cookie-based login system, I added that to my site for improved security. When an employer taught me about C# webmethods, I reconfigured existing functions in my site into webmethods so that they could be called seamlessly from javascript, instead of using postback methods. When I started working on collaborative team projects with other programmers, I was shown the benefits of using git repositories for backups and progress tracking, so I moved my site into one. At my most recent workplace, I was faced with collecting user/stakeholder feedback and incorporating it into a project plan, so I designed a ticketing and project management system for them, a version of which I now use in my personal site. Whenever I code new systems, I try to start with generalized building blocks, functions that can be used for a variety of circumstances, so I started creating classes of related functions that can be shared across my projects, such as generalized functions for searching and sorting through List objects, or safe (non-breaking) functions for parsing/converting Strings to other formats. Every piece of advise I received, or mistake I made, or lesson I learned, was incorporated into my code moving forward. This ensures that the logic and structure of my code are as modern and efficient as possible.

This approach highlights my flexibility as a programmer, and my willingness to not only learn as much as possible, but to recognize my mistakes and improve upon them. As programmers, we should always be seeking to improve our work, and I took this opportunity to implement some of the lessons I’d recently learned but hadn’t yet had the opportunity to build into my code. These three enhancements, described below, revolve around the concepts of **software design & engineering**, **algorithms & data structures**, and **databases**. For the software design & engineering component, I chose to address a matter of a poor user experience using a clunky, outdated system. This showcases my understanding of the final user experience and drive to design clean, useful web tools, as well as my profinity for updating old code to newer languages and structures. For the algorithms & data structures update, I improved the efficiency and usefulness of a code-generating tool. This ultimately showcases my aptitude for efficiency, and designing systems with that in mind. For the database portion, I implemented more advanced concepts of MSSQL that improve the efficiency & security of my auditing system. This showcases appreciation for data integrity, which is vital for any programmer, and my awareness of auditing requirements across various industries.

All code for this project can be found in my GitHub Page, https://github.com/dillonpuglisi/Capstone/blob/gh-pages/index.md

**Enhancement One: Software Design and Engineering**

The documents for this enhancement are the front and back end code files for the Notes page of my personal management site. Through this page, the user can create, view and edit notes records. Statistical information is displayed in feeds adjacent to the add/edit form, showcasing the most recently added/updated notes records, most frequently viewed, etc. This was one of the first pages created for my site a few years ago, and as such, features the oldest and most poorly structured code. There’s very little commenting on my functions, and they employ older, less efficient logic in their execution. Originally, the note add/edit functions were postback methods, meaning that for them to be run, the whole page gets refreshed, which is outdated and disruptive to the user.

For this enhancement, I focused on updating the add/edit note functions from postback methods to webmethods. The server-side elements in the front-end page had to be updated such that their values could instead be read by a front-end javascript function, which would process those values through an AJAX call to webmethods in the backend. The backend C# code would then add or edit the note record and report success or failure to the javascript function, which would then have to update the page accordingly. With this, all the functions that refresh the feeds on the page also had to be converted to front-end webmethods so they could be called by javascript. I also wanted to take this opportunity to more sufficiently comment the related functions, make use of newer generalized functions I’ve recently developed-- such as a custom javascript function that runs AJAX calls for me, given basic inputs such as the webmethod name and input variables object-- and improve the IDs of my elements to be clearer in their intended use. All of these changes would update the status of this system from personal project to a more professional system, showcasing my skills in developing clear, logical systems with regard for both the end users’ experience, a programmer’s ability to return to and update the code at a later time, and system security.

Though I did not go through and update every function in the codebehind, I did improve and clean up all the functions related to adding or editing a note record, such as AddNote() and UpdateNote(). The only real difficulty came from using functions that I’ve developed mostly at work, and haven’t completely implemented in the code of this project. For example, the javascript function I wrote at work that updates the content of an element for me, given minimal input, makes use of a loading icon gif that I haven’t gotten around to uploading into the project files of my site. So while the update function works, the loading icon doesn’t properly display while it’s loading the changes. To resolve this, I will have to copy the file the next time I’m at work and upload it here. Similarly, I’ve improved some of my generalized functions at work, and haven’t yet copied over those enhancements to those functions here, so I will not able to make use of some of them until I get the updated code from my work computer. Regardless, the updates to the system still work and work well. Note creation/updates are fully handled by the front-end, resulting in a much more seamless experience for the end user and a much greater value to me as the owner. The code is cleaner and more condenced, with more commenting for clarity-- this will also make it easier to implement further revisions down the road. I also included some security-related updates, such as access level restrictions on vital functions like the add/edit note methods-- which would prevent unauthorized users from modifying vital data-- but they were commented out since this site doesn’t yet permit multiple users on it. More functions were wrapped in try/catch statements to improve system integrity.

One of the biggest things I learned throughout this process was the importance of doing things right the first time. Going through and commenting older code is harder when you haven’t worked on it for years. It also reminded me of the importance of using version control; I put this week’s changes into a separate branch off of my main project, which became useful when I needed to roll back some changes due to missing code, as described above. Previously, I would’ve simply made the changes directly in the main branch, and once I realized code had to be rolled back, I would’ve had to dig up an older copy of my site and manually revert the code back by coping and pasting it from the backup. Also, I was reminded of the value of more fully planning out code revisions before getting started. I didn’t realize until I delved into it that if the note creation would instead be handled in the front end, and the page has to redirect to the new note after creation, then the new note ID has to be provided by the AddNote function and processed back into the javascript function that called it. This required me to alter that function beyond my initial revisions in order to accomplish that.

**Enhancement Two: Algorithms and Data Structure**

The documents for this enhancement are the front end code files for the WebTools page of my personal management site, an .aspx file and a javascript file. Through this page, the user can run any number of operations that make writing coding easier, such as through a simple code formatting tool, a data parser, or the subject of this enhancement, a code generation tool. This page was created more recently in the development of this site, so it’s commenting and code structure is a bit better than the focus of my last enhancement, but could still afford some improvements. The separation of tasks amongst functions is efficient, but the code generation tool needed a small overhaul to better serve my needs.

For this enhancement, I focused on updating the system that writes code for me, given the name and fields of a table in the database, along with some other basic inputs. The inputs are processed through various javascript functions that parse the object and field names into several blocks of C#/JS code, comprising a basic management system for that table, that I can copy and paste directly into projects. The functions it generates include methods for retrieval of data from that table, a class object and container object to store that data in, a function to process that information into a formatted html table for display, various functions for adding/editing/deleting records, and other management systems. Since most of the database management systems I build for myself or my job follow this basic format, this system has saved me considerable programming time, but it is not as versatile as I would like. There are some common cases that I would like it to cover, such as when the data retrieval function should get its information from a view rather than the table directly, or when the pluralized version of the object name does not fit with the logic as its currently coded, and should thus be specified by the user. This requires some additional input fields. Additionally, since fields possess different types and should be handled differently, it needs an intermediary step that allows the user to specify different customization options for each field. This will require the addition of a Preprocess button to run a new function that will take all the fields and generate a form for these customizations, and then those additional inputs must be processed into the functions that write the code. This includes a specification of which field is the primary key, which fields should/should not be editable, and which datatype represents the field, so the AddEdit form code it builds knows how to design the <input> elements for those fields. These will showcase the skills/principles of complexity and versatility, delivering value and helping me better serve my job as a developer for my machining company employer, who requires innovative and efficient solutions for our industry-specific web tools. It will allow me to write code for our numerous systems much faster and more efficiently, as I will then have to modify/customize the code generated by this system less to fit the structure it’s applied to.

Though I did not go through and update every function in the script file, I did improve and clean up all the functions related to this specific system, such as BuildAll3() and the functions run therein. I fixed some formatting issues with the form while I was there, such as the final BuildAll btn being dragged down by the output textarea when it should always be above it. I was also able to add the additional, optional fields for the viewname and pluralized objectname and thread them into the build functions. If they are left blank, the system proceeds as normally. For the field customization options, I was able to build the preprocessing system and input form, and confirmed that these new fields are properly detected by the system at the time of the final Build.



Now, field modification is bound to its datatype and declared editability, along with any additional refinements set by the user (such as number min and max). For an additional change that was not part of the original plan, I thought it would also be good to establish defaults for the the field customizations, to make that process faster, so the first field is initially presumed to be the primary key and all fields are marked editable by default. I also realized that many of the fields I deal with are bits/booleans, so I realized late in the process that I would benefit from adding a bit/bool datatype to this system. If a field is marked as such, the input datatypes and the manner by which the data is processed from the SQL table to the container object too has to be modified, using custom functions I’ve built to cleanly handle conversions of that datatype.

The biggest complication arose with a feature I considered adding-- the ability to specify the table that foreign key fields reference-- but in application, that proved too difficult to implement effectively, so I did not include it. All other planned updates work and work well. The code generation system now grants me much greater flexibility, allowing for full, more complex data control systems to be generated instantly, requiring next to no manual customization for more unique structures. More comments were also added to the functions for clarity-- this will also make it easier to implement further revisions down the road. More functions were given data integrity checks to ensure that nothing breaks if it attempts to run operations on empty inputs. I also extracted out the creation of container names into the main function so that could be passed into each function that uses it, rather than having to recreate that logic in each function.

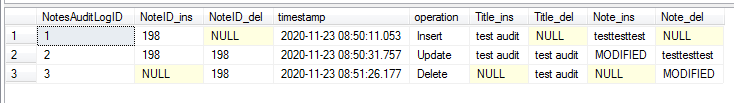
This modification reiterated to me the importance of doing things right the first time. Since my commenting was 50-70% sufficient already, making the modifications was much easier, and certainly less time was required to add more comments. I’ve worked on this system more recently, so the process of refamiliarizing myself with it was quicker. As with the previous modification, I used version control to keep these changes separate just in case I wind up needing to roll them back or implement a custom merge of changes.

**Enhancement Three: Databases**

The documents for this enhancement are the SQL script that modifies the trigger I created for the dbo.Notes table, and some screenshots demonstrating its functionality. As described in a previous modification, the Notes.aspx page was one of my earliest created a few years ago, allowing the user to create, modify and delete notes files. In order to log the progression of changes made to a record, the system manually creates an audit log through a C# function that gets called every time an insert/update/delete function is run on that table in C#. This is clunky and a poor separation of tasks, requiring that a call to that function be attached to any function added to the site that modifies the notes table. As such, and to employ techniques learned recently from my job, for this modification, I created a trigger attached to the notes table itself that would itself create a record in a new audit table any time a note record is added/changed/deleted. This should also make those operations run a little faster from the front end, since the audit logging is no longer handled by C#.

I had to first create a new table to store these logs-- dbo.Notes\_AuditLog-- making sure to include all relevant dbo.Notes fields within that table, such as the timestamp, operation performed, and both a Title and Notes field for values before and after the operation is run. Once that was done, I was able to build the trigger itself. Using SQL Server management studio allowed me to make use of a ready-made template which outlined the structure of the procedure. One of the biggest challenges I faced was determining what operation was being performed, but after some research and testing, I found an algorithm that would handle that for me. I made sure to sufficiently comment the procedure and name all variables in a manner that was descriptive. I considered developing a pair of tables that would be capable of storing audit records for every table-- with the parent table recording the top-level information about the change, what table it was, when the change was made, etc, and the related child table listing all the fields changed under that operation-- but this proved difficult to implement quickly, so I’ll have to develop that a bit further if I decide to try that out in the future. It would be a useful tool that I could apply to my professional work, I would just have to design a system of generalized procedures that could insert audit records into these generalized tables under any table in the database. This would require looping through all the fields in those tables and querying the inserted/deleted tables using those field names, which experience has taught me is tricky.

To confirm that the trigger works properly, I created a new note file, then modified it, then deleted it. The resulting audit logs sufficiently capture these operations, although since my database is hosted in another timezone, I still have to deal with GETDATE() returning the timestamp from a different timezone. I still haven’t quite figured out how to resolve that.



This technique of audit logging can be applied to my professional work, as it is not something that our system at work currently handles. To be compliant with regulations, we need to do a better job of logging who changes what records and when, and applying this system will accomplish that. Doing so will improve the integrity of our data, plugging the potential vulnerabilities in our system and accomplishing industry security goals.