

NORTHWEST NAZARENE UNIVERSITY

Manuscript and Comment Creation and Viewer Software Development Project

THESIS

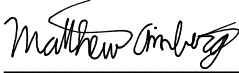
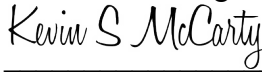
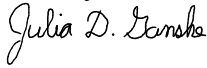

Submitted to the Department of Mathematics and Computer Science  
in partial fulfillment of the requirements  
for the degree of  
BACHELOR OF SCIENCE

By  
Matthew Amberg  
2023

THESIS  
Submitted to the Department of Mathematics and Computer Science  
in partial fulfillment of the requirements  
for the degree of  
BACHELOR OF SCIENCE

By  
Matthew Amberg  
2023

Manuscript and Comment Creation and Viewer Software Development Project

Author:	 <hr/> Matthew Amberg
Approved:	 <hr/> Kevin McCarty, Ph.D, Department of Mathematics and Computer Science, Faculty Advisor
Approved:	 <hr/> Julia Ganske, Ph.D., Department of Music, Second Reader
Approved:	 <hr/> Barry L. Myers, Ph.D., Chair, Department of Mathematics & Computer Science

# **Abstract**

Employing Full Stack Development to Create a Viewer for Manuscripts and Commenting Systems.

AMBERG, MATTHEW(Department of Mathematics and Computer Science), MYERS, DR. BARRY (Department of Mathematics and Computer Science), MCCARTY, DR. KEVIN (Department of Mathematics and Computer Science).

The overarching goal that passes the scope of this project is to offer a site for writers to upload their manuscripts and to receive criticisms as well as offer criticisms to others' works. As the site was very early in the works, there was a need for a baseline system for being able to store those documents, view them, as well as a robust system for being able to create comments. A fullstack methodology was employed in the creation of this system. Three-Tier Architecture is the structure of the application and React, Javascript, and ASP.Net are the tools used.

The main goals within the scope of this project are to allow a user to highlight and visually see what they selected and enter a comment. Additionally, the goals included the capacity to select pre-existing comments, being able to read the comment left, as well as reading and entering additional sub comments to a head comment. The project set a baseline for further development of the site with future work and projects.

## **Table of Contents**

<b>Abstract</b>	<b>iii</b>
<b>Table of Contents</b>	<b>iv</b>
<b>Table of Figures</b>	<b>iv</b>
<b>Introduction</b>	<b>1</b>
<b>Background</b>	<b>1</b>
<b>Implementation</b>	<b>2</b>
<b>Results</b>	<b>11</b>
<b>Future Work</b>	<b>12</b>
<b>References</b>	<b>13</b>

## **Table of Figures**

Figure 1. Three Tier Application Architecture	3
Figure 2. Five project layers in Three Tier Architecture	6
Figure 3. Uploading Documents through the Five layers	7
Figure 4. Screenshot of Manuscript Viewer, Comment Box, and Highlighted Text Examples	9

## **Introduction**

Writer's Klatch is a website in the making that a portion of work was set aside for this project. The website is intended to be a place for writers to store, share, and critique each other's stories with many tools to allow for ease of use and functionality. One such functionality that was the goal of this project was the baseline viewer of manuscripts, and the functionality to select text, add comments and view the comments.

Some sites offer this sort of functionality, such as Google Docs, but the tools in regards to controlling viewability of comments, comments on different manuscript versions and other specialized tools are not available in Google Docs. Google docs is a more generic document storing and editing tool with its collaborative features. Some tools that writers might be looking for such as versions of documents and more robust commenting functionality are not readily available in Google docs.

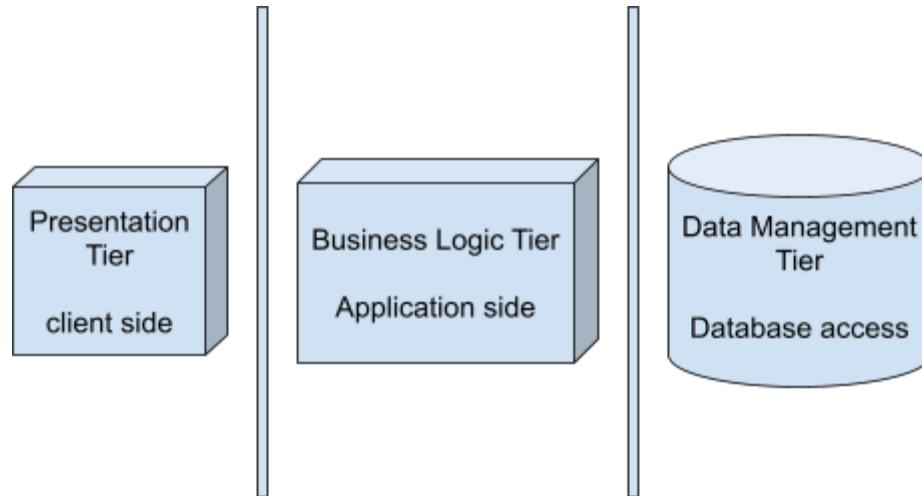
## **Background**

Some writers intend to write books and publish them. Others simply write as a hobby and simply wish to share their works with others. Some tools that have allowed for that are document storing and editing tools such as Google Docs. A problem with sites like Google Docs is that those sites are more generic in design. They allow for multiple sorts of uses of document editing, such as school assignments or journal publication documents. There are other sites that are more specialized for sharing and critiquing of manuscripts but they do not have as robust of a critiquing system as this project's final goal intends to have. The company Scrumfish has its site, Writer's

Klatch, in the works and this project was scoped to adding said functionality to the preexisting site.

## **Implementation**

The project was developed using ASP.Net and React in a full stack environment. This meant that development was done primarily in ‘Three Tier Architecture,’ which in this project consists of 5 notable layers. Three tier architecture is an application development standard meant to encapsulate data and information to control access to said data and security of data being sent to and received from the site. There are three tiers, as can be seen in figure 1. The top is the presentation tier, which contains the code for things seen by the clients. This front-end code is usually in some form of HTML or Javascript. The second is the Business Logic Tier. It handles the incoming and outgoing data. This tier is what the presentation communicates with but is not inherently visible to the clients, which ensures the security of the site's inner workings. Finally there is the Data Management Tier. This lowest tier primarily handles the storage of data and is responsible for returning data to the Business Logic Tier when requested and storing what is sent to it from that tier.



*Figure 1. Three Tier Application Architecture*

While there are three primary tiers, this project was structured in five separate layers that fit these given tiers as shown in figure 2.

The lowest layer was that of the backend database and data storage layer. This layer primarily consisted of SQL database tables and Azure blob storage. This layer functioned primarily as storage and was not directly altered at that layer itself outside of their initial creations. This layer was altered and used by the next layer up, that being the DAL (Data Access Layer). Within the DAL is a collection of entities with various attributes. This collection of entities is what implicitly creates columns for the SQL tables and updates them when a database migration is run. The functional code in the DAL consists of files labeled with their appropriate data access ending in 'Data,' such as the example of the file labeled 'UserData.cs.' The code in the DAL takes inputs that have come down from the top layer down to the business layer, and adjusted for entry to the data storage through instantiated entity objects. These two lowest layers comprise the Data Management Tier.

The BIZ (Business) Layer is the layer that takes data from the upper layer and manages it. An example of that being the code for registering a new user, the upper layer simply passes down the input credentials. The BIZ layer is the one that hashes the password before it is passed to the DAL for storage. Additionally, for the sake of encapsulation, DAL's have various coded functions, such as to check for duplicate user emails as an example, but do not do said logic in the same function as the function meant to upload the input adjusted data from the BIZ layer code that called that function. The BIZ code uses the various DAL functions and has the logic to catch any issues with the given code. The DAL functions as the tools to store the data.

(BIZ layer pseudo code)

Login Function (UserCredentials User)

    If DAL.LoginUser(User.Email, User.Password.Hash(), User.MFACode)

        Return Ok(UserModel {

            Email = User.Email,

            LastLogin = DateTime.now() }

    Else

        Return Unauthorized()

End Login Function

Both the BIZ and the DAL objects were created through interfaces and instantiations of those interfaces. This was done to follow best practice and create abstraction of the code. Such examples of abstraction were the creation of the



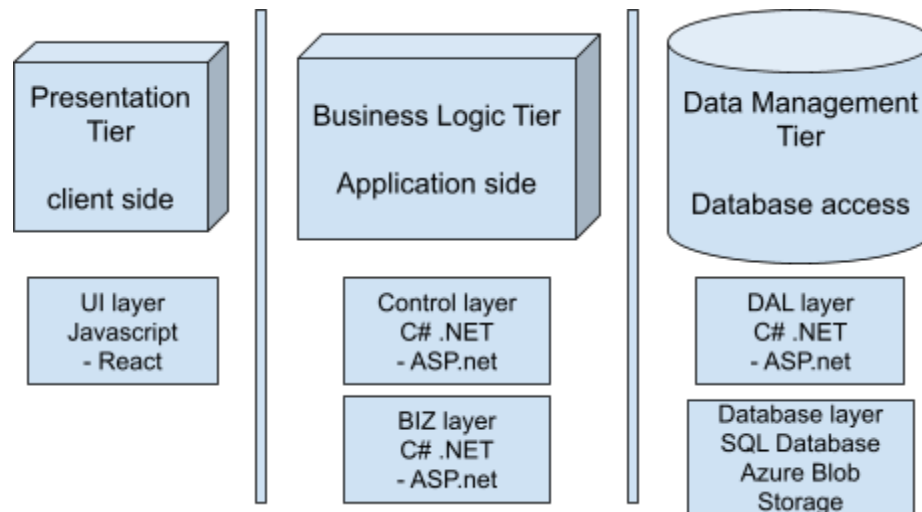
‘IManuscriptData’ object which was the interface for the actual ‘ManuscriptData’ object in the DAL. The interface simply had entries for each function, while the DAL object had the code for each function. All BIZ objects and DAL objects are structured in this manner.

The next layer, labeled as the ‘UI’ layer is the layer in which the api controllers are located. The controllers are functions that the front end code calls and inputs raw data into. The controllers receive the input from the frontend and instantiate a model object to pass the data down to the BIZ layer for the BIZ to handle it and the logic for it. There is some logic done on the Control layer, but for the most part the Control layer formats the input data into a data model for the BIZ layer to further format and apply logic to.

Additionally, while not a layer, there is a container of ‘objects’ that is referenced in the BIZ and Control layers. Objects such as the ‘user’ or ‘comment’ objects, have models for passing the data between the layers. One such example of this is the ‘UserModel’ which is a data structure that the Control instantiates to pass the input user data (including password) to the BIZ for the login logic. This model is good for passing the user data down, but there is no need to pass some extra credentials, such as the user’s hashed password, back up when returning user data to the UI layer. As such, there is another model named the ‘UserDisplayModel.’ This is similar to the UserModel, but it does not have the extra attributes, such as the password, or other sensitive information that was input initially for the login logic. This model is passed back up by the BIZ to the Control for the Control to return to the UI layer, which then displays the information in some format. The BIZ and the Control layers comprise the

Business Logic Tier. These two layers handle incoming data, formatting, and applying the business logic to said data, as well as formatting and returning data sent up from the Data Management Tier.

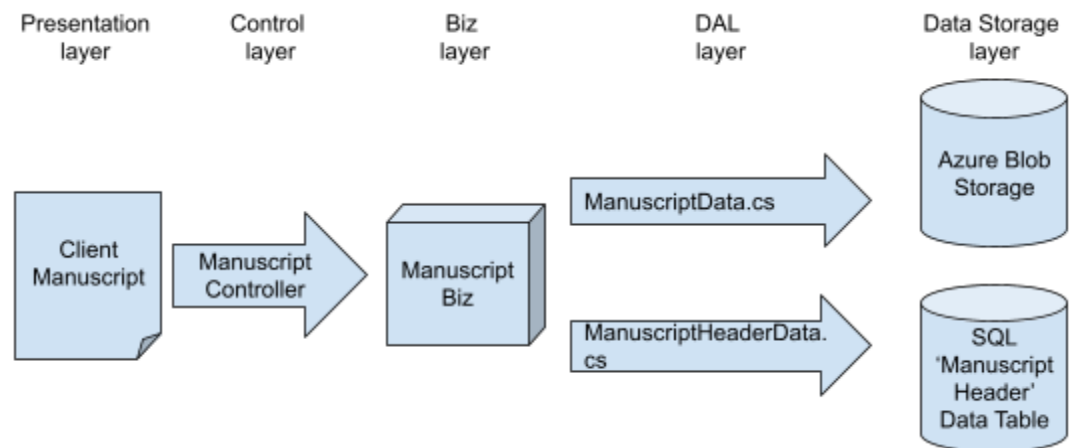
The top layer is the UI layer. This layer is done in Javascript with React and has a lot of its own logic and subdivisions in the code files. This layer is the code for what is actually presented on the screen, the logic for displaying user data, moving between pages, and calling the Control layer functions. The Control layer functions are given routes and http functionalities. Such as Http Get or Http Put. As such, the UI layer also handles taking the user inputs and formatting them into a json structure and inputting them into the http fetch calls to pass the data to the Control layer for it to format and handle. The UI layer itself comprises the Presentation Tier.



*Figure 2. Five project layers in Three Tier Architecture*

The functionality to upload documents to the site and to store them was the first major goal to be completed in this project. Figure 3 gives a top down layout of the code structure. The code for this was started in the Control layer. A control was created with the primary function intended for this first task. The primary function consisted of an

HTTPPost method, taking one parameter, an IFormFile, which is the byte stream and other metadata of the uploaded document. This function passes the file data, the file title, and the user ID to a BIZ object that handles manuscript information. The BIZ then takes the header data and passes it to a DAL that is connected to a table for header data of the manuscripts, storing data such as the user ID, the manuscript ID, and the creation date. It also stores the version number as users are allowed to upload multiple versions of the same manuscript for when they make changes. This BIZ also passes the file data itself to another DAL. This DAL's connection is to the blob storage, which stores the files for the documents themselves. The DAL also applies IDs to each character for use in the UI layer, and then saves each page individually to the blob storage. The UI layer deals with accepting a file to upload and handles entering it through a fetch question for the httpPost function.



*Figure 3. Uploading Documents through the Five layers*

The next task was creating a system for viewing the document and selecting text to create comments. A screenshot of this is shown in figure 4. The viewer was created in the javascript UI layer. The other layers all simply grab the data from the database and blob storage and pass them up, formatting them into json objects which then the UI

layer can populate the page with. The webpage itself first loads up the first page of a selected document. The url was ordered by the document title, the version, and the current page number. The fetch functions with the httpGet were structured similarly. A pagination library was used to alter the current page variable which was what was kept in the url and sent through the fetch call. This meant the page would load a new page of the document with each page change, rather than having the entire document all loaded at once.

Due to the ID'ing of each character in the document, those IDs could be obtained and compared to highlighted selections of text in the document. Doing so, the code recorded the first and last ID of the selected text. A textbox was coded on the page to be placed next to the document viewer. The comment box had a button that was disabled if there was no selection. If the selection was valid and the textbox itself was not blank, clicking the button activated another fetch function which inputs the comment text, the manuscript ID, the user ID, and the first and last element IDs of the selected text. Those are then passed through the layers down to a new DAL which stores the comment in their own table. Additionally, if the selection includes an already existing comment, the selection IDs are not passed and the parent comment ID is passed to the Control. The parent comment ID is an optional parameter when input, which allows the control to determine whether the received comment is a subcomment or a parent comment. It then calls one of two BIZ functions to pass down the respective comment or sub comment.

*Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris diam nunc, venenatis ut lectus sit amet, blandit malesuada tortor. Maecenas pharetra tincidunt nisi aliquam cursus. Duis vestibulum diam id augue mattis sodales. Nam ipsum purus, dictum eu tellus et, ultricies rhoncus erat. Donec nec nunc malesuada, fringilla neque eget, laoreet turpis. Ut accumsan placerat neque quis bibendum. Interdum et malesuada fames ac ante ipsum primis in faucibus. Donec molestie egestas aliquam. Mauris tristique, turpis sed malesuada condimentum, quam leo suscipit nibh, pulvinar bibendum magna ante a nibh. Sed sed commodo urna, eu ultricies lacus. Sed porttitor sem hendrerit, laculis ante sed, suscipit massa. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Vestibulum congue tellus ut mauris pulvinar laoreet.*

Quisque sed viverra justo. Curabitur bibendum porttitor neque vel ultricies. Suspendisse fringilla finibus mi, at lacinia eros pharetra ut. Phasellus consectetur nibh ac tempus fringilla. Donec interdum ultrices dui. Vestibulum quis urna lectus. Fusce at porta erat. Integer ornare magna a purus convallis, nec vehicula eros fermentum.

Pellentesque at enim eros. Aliquam erat volutpat. Mauris id sem sit amet tellus placerat volutpat at non enim. Suspendisse velit odio, rhoncus nec velit accumsan, sagittis finibus turpis. Aliquam nibh urna, lobortis sed cursus vitae, accumsan ut quam. Fusce facilisis felis ut quam molestie, id posuere nulla accumsan. Aliquam tempor lorem pharetra, tempus metus eget, lacinia mi. Nulla facilisi. Morbi hendrerit ipsum eu sem rhoncus auctor. Vivamus bibendum lacinia condimentum. Aenean faucibus suscipit ipsum, quis molestie mi faucibus ac.

Mauris nec nisi elementum, facilisis purus et, mattis lacus. Maecenas eu blandit urna. Cras placerat vitae neque a dignissim. Aenean risus orci, dapibus sit amet nisi a, cursus aliquam urna. Donec in pharetra lectus. Nam vitae leo eu mauris euismod egestas. Donec sed lorem a quam fermentum viverra. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Ut lobortis interdum maximus. Curabitur eu fringilla justo. Cras sagittis dolor ac sem semper, sit amet sollicitudin sem viverra. Vestibulum et tellus mi. Quisque accumsan ante arcu, a convallis nunc efficitur sed. Maecenas aliquam, orci id mattis hendrerit, tellus tellus laoreet tortor, sit amet bibendum lacus lectus ut ligula. Nunc scelerisque non lorem sed mollis.

Integer viverra metus odio, eu faucibus lacus posuere vel. Curabitur non vestibulum velit. Aliquam maximus tortor id ipsum malesuada scelerisque. Phasellus facilisis mattis turpis. id euismod velit

Enter comment:

orange selected

Send

Figure 4. Screenshot of Manuscript Viewer, Comment Box, and Highlighted Text

### Examples

The UI layer handles the fetch calls and each UI layer functions to load data received from fetch calls in an ordered way. The Javascript page was separated into 3 primary components. There is the manuscript viewer, which is a div with the html inset into it. The comment viewer, which is a div that is populated with the text from loaded comments, and the comment box, which is an input box and a button which is used to send comments to the database. Due to the asynchronous nature of the Javascript variables, these components all were loaded in a specific order, as each component loads more data which is contingent on the prior data being loaded. If they are not loaded in order, null exceptions occur which would break the page.

When actually loading things onto the page in html style (not applying functional logic to format data), there is some simple boolean logic that is done to control when things are loaded. For the manuscript viewer, all the components, including the div for viewing the document, the comment viewer, and the comment input box, are barred from being loaded through a boolean statement. The boolean statement is as follows:

Manuscript Viewer Code

If HTML Document Byte stream not null

And If Comments have been pulled from the Application Tier

Load html and react components

Populate variables on page with input and loaded data

End Manuscript Viewer Code

The comment viewer is an empty list that can be populated with text. It loads the stored comment of each highlighted existing comment on the page and presents it to the user.

The comment input box is the other important component. As it is only instantiated onto the page after the document itself is loaded, it then has functions in it to determine the IDs of the ends of the loaded manuscript page and highlight existing comments on the page within that range, as well as allowing for the user to type a comment and create it by clicking the send button, which then sends the text for the comment to the controller.

## Results

The app prior to this project was a mostly base web application with minimal functionality. After the completion of the project the site gained login functionality as well as the functionality to upload multiple versions of manuscripts, view them, leave comments on them, and view said comments and see what portions of the text were highlighted where said comments were left.

The ease of use of the added functionality from the scope of this project are fairly streamlined and straightforward. Selecting a portion of text to highlight is as easy as highlighting it with a user's mouse. Changing a selection is equally as easy as highlighting a new or different portion of text.

As the documents are not intended to be edited by viewers, there is not a method to step through each character with a cursor like in a document actively being typed in, so viewing comments is done in a similar fashion as selecting comments to add text. Simply highlighting any portion of a preexisting comment will load the text of the comment into a comment viewer. Highlighting multiple existing comments will load all of them into a table to view.

Finally, there was also code on most of the layers to ensure that any possible overhead with the system is addressed. Users are unable to click the send button if they have not selected any text in the manuscript or have not typed any non-white space characters into the comment box. The presentation layer also will not attempt to call the controller if a function is called with empty text or no selected text.

The method of uploading and viewing manuscripts is straightforward as well. There is an upload dropbox that you can either drop a file into or click to find and

upload a file. It only accepts .docx files, so there cannot be any issues in attempting to upload an improper file type. The dashboard page which has the upload dropbox also has a list of all manuscripts for a user as well as the highest version number of each manuscript. This gives a quick look at all the manuscripts a user has, how many versions each manuscript has, and an easy way to traverse to the viewer for each manuscript.

## **Future Work**

The scope of the project was limited to the fundamental functionality of uploading and viewing manuscripts, as well as leaving comments and viewing the comments, and the functionality around highlighting and selecting text for commenting.

The main goals of the project were; a) create a viewer for documents, as well as code the functionality to upload and view manuscripts, b) create functionality to highlight text in a document in real time for selection, and c) code a method to send comments for selected text, and be able to see what comments were created as well the text selected where the comment was left, and the content of the comment. These goals were all accomplished, but there is more work to be done. One such thing to be done is the necessity to account for viewing manuscripts uploaded by other users. Currently the lower layers of the code can account for users when grabbing documents to view, and when uploading and leaving comments, but the Control layer does not have any methodology currently for grabbing any other user ID than the one of the user currently accessing the site. This means that currently a user can only see and access manuscripts they uploaded themselves and therefore have their user ID attached to the manuscript in



the header data table. Due to the layered nature of the project, the only changes that would need to be made would be at the UI layer to be able to pass other user IDs rather than just the ID of the current client user.

Other future work that is required is the presentation aspect of the UI. The scope of my project was primarily functionality over visual stylization. As such, all the data being passed up from the Control layer to the UI layer is adequate and available for use, but the way it is presented is fairly minimalistic and barebones. Cleaning up the UI layer to create a more streamlined and visually sleek experience for the user is future work for this project.

Additionally, the other aspects of the site, such as a means to browse uploaded manuscripts by various users, user accounts and profiles, and more are all other large tasks that are all future work that was outside the scope of this particular project.

## References

- Earthpledge Foundation. (n.d.). *Three-tier architecture overview*. Amazon. Retrieved April 25, 2023, from <https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/three-tier-architecture-overview.html>
- Forms*. React. (n.d.). Retrieved April 25, 2023, from <https://legacy.reactjs.org/docs/forms.html>
- Rick-Anderson. (n.d.). Routing to controller actions in ASP.NET Core. Microsoft Learn. Retrieved April 25, 2023, from <https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/routing?view=aspnetcore-7.0>