<Entitled> Website Project Report

Alex Stack – CSC 481 Systems Analysis

ASP.NET Web Application and Microsoft SQL Database

12/08/2019

**Summary:** A web application that serves as a central portal for a video game organization (aka guild). This website will serve as a welcome site for the guild, containing a welcome letter, guild information, and a user portal. The website is being made due to the organizational requirements to operate a large-scale guild within the game, needing a way to keep track of member participation and shared resources. A majority of the website will be a guild bank: this bank will be a record of all the resources gathered by the guild, allowing members to donate and request items. The website will also keep track of participation points, called DKP, that members will gain by either donating resources or completing guild events and spend in the guild bank or on valuable items rewarded by guild activities. The website will be constructed in Visual Studio using Microsoft’s ASP.NET core, implementing a Microsoft SQL database. Cloud computing is also being investigated at this time, using Microsoft’s Azure framework to host both website and database, with the potential of developing an API to integrate the website with a separately developed in-game Lua application.

**Proposed Work:**

User must be able to:  
 1. View guild information including roster, activities, and bank  
 2. Sign up and create unique user profile  
 3. Request to donate or take items from guild bank  
 4. Complete a participation form to gain DKP  
 5. Keep track of DKP, spend DKP on bank items, spend DKP on raid items  
 6. Admins must be able to view invoices and approve/deny them  
 7. Users can create invoices, which must be reviewed by an admin  
 8. Admins can add and remove new items to bank

These are the core actions that the users must have. In addition, with the creation of a web API an in-game addon may be created to automatically record participation in guild activities and points spent on raid items. The website may also be hosted on a cloud-based platform, depending on licensing and timeframe.

**Initial Workflow Plan:**

Stage 1: Initial plan created, conception, finish choosing technologies for project

Stage 2: Finalize all details, technologies, site layout and style

Stage 3: Create skeleton of website, front end with user login control

Stage 4: Create invoicing system and participation form

Stage 5: Create database front-end for CRUD invoices (sort, search, approve, deny)

Stage 6: Finish guild bank and create web page for displaying bank (sort, search)

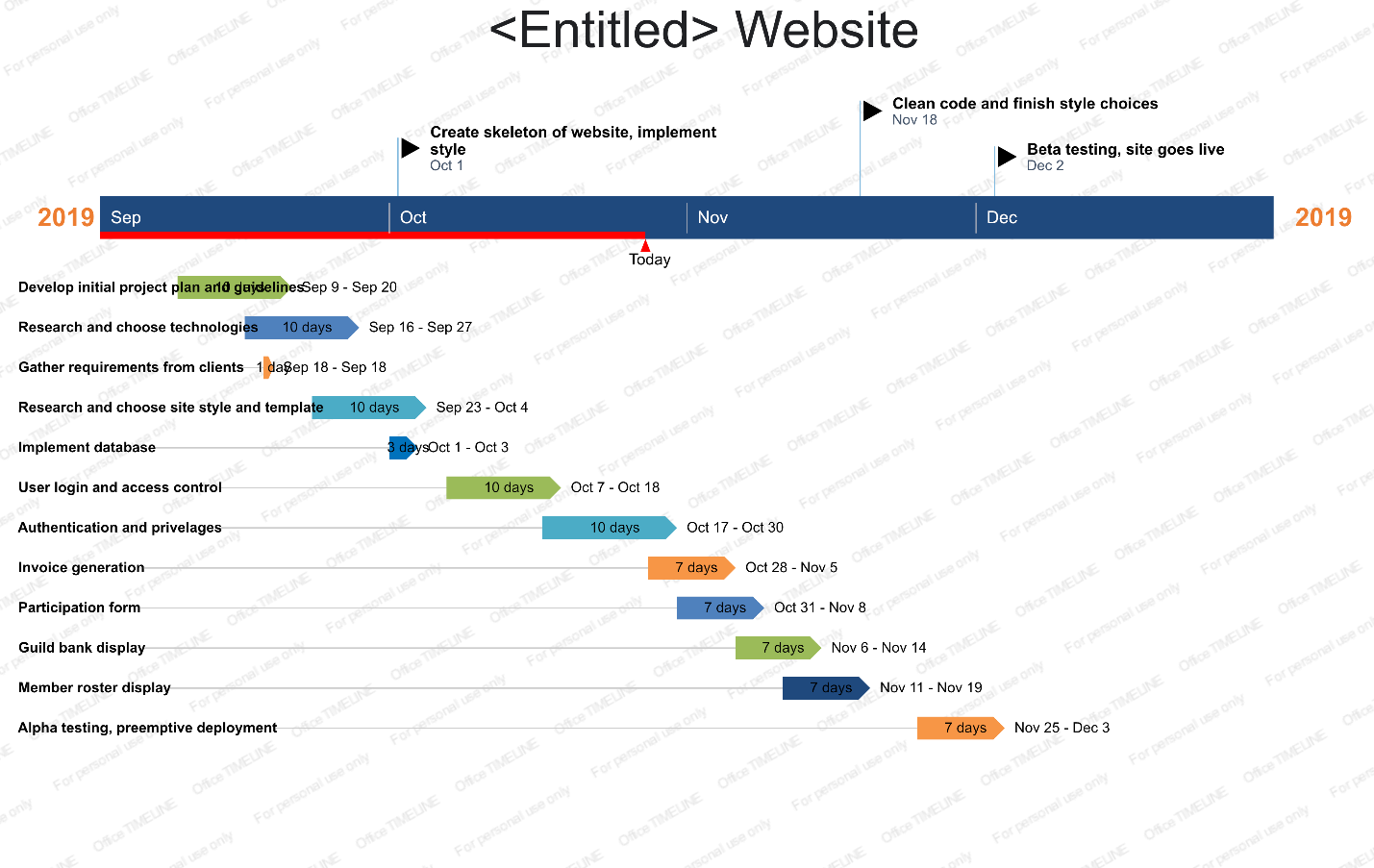
Stage 7: Clean code, beautify website, begin alpha testing.

Stage 8: Deployment, beta testing (by guild members), finish reports.

Optional stage 9: Export database to cloud, deploy site to cloud

Optional stage 10: Make web-API for in-app integration

Optional stage 11: Develop in-game application to link to site



Most of the time spent developing the website was learning .NET. My previous experience had prepared me for the C# coding, SQL commands, and an understand of basic web development. However, learning the MVC paradigm took the most amount of time, as it is a concept that was previously unknown to me.

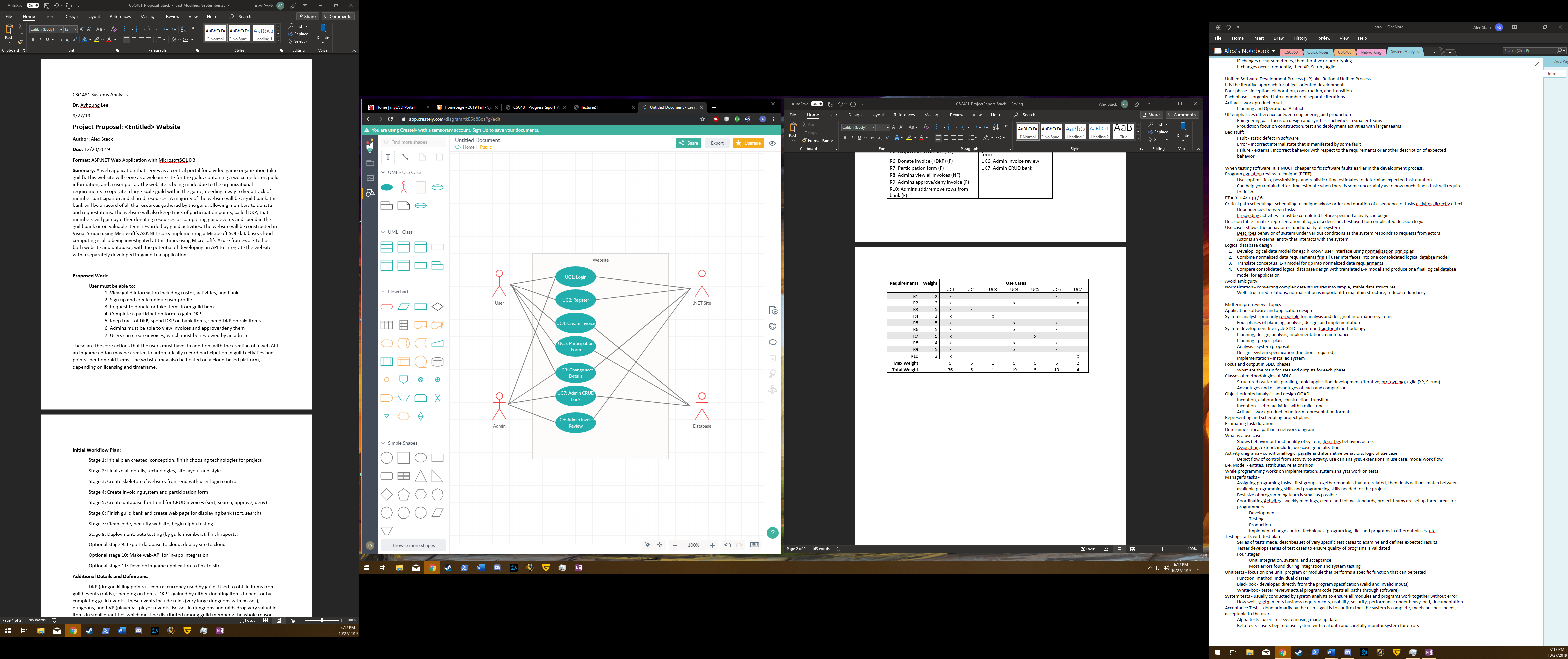
**Project requirements, functional and non-functional:**

|  |  |
| --- | --- |
| **Requirements (Functional and non-functional)** | **Use Cases** |
| R1: View roster (NF) | UC1: Login |
| R2: View bank (NF) | UC2: Register |
| R3: Register user (F) | UC3: Change account details |
| R4: Modify unique account (F) | UC4: Create invoice |
| R5: Request invoice (-DKP) (F) | UC5: Complete participation form |
| R6: Donate invoice (+DKP) (F) | UC6: Admin invoice review |
| R7: Participation form (F) | UC7: Admin CRUD bank |
| R8: Admins view all invoices (NF) |  |
| R9: Admins approve/deny invoice (F) |  |
| R10: Admins add/remove rows from bank (F) |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Requirements** | **Weight** |  |  |  | **Use Cases** | |  |  |
|  |  | UC1 | UC2 | UC3 | UC4 | UC5 | UC6 | UC7 |
| R1 | 2 | x |  |  |  |  | x |  |
| R2 | 2 | x |  |  | x |  |  | x |
| R3 | 5 | x | x |  |  |  |  |  |
| R4 | 1 | x |  | x |  |  |  |  |
| R5 | 5 | x |  |  | x |  | x |  |
| R6 | 5 | x |  |  | x |  | x |  |
| R7 | 5 | x |  |  |  | x |  |  |
| R8 | 4 | x |  |  | x |  | x |  |
| R9 | 5 | x |  |  | x |  | x |  |
| R10 | 2 | x |  |  |  |  |  | x |
| **Max Weight** |  | 5 | 5 | 1 | 5 | 5 | 5 | 2 |
| **Total Weight** |  | 36 | 5 | 1 | 19 | 5 | 19 | 4 |

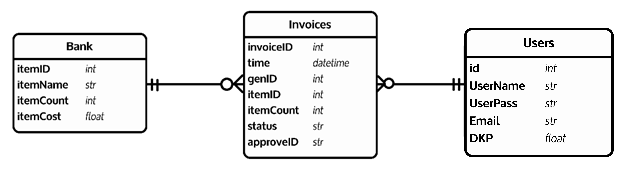
**Project Use-case diagram:**

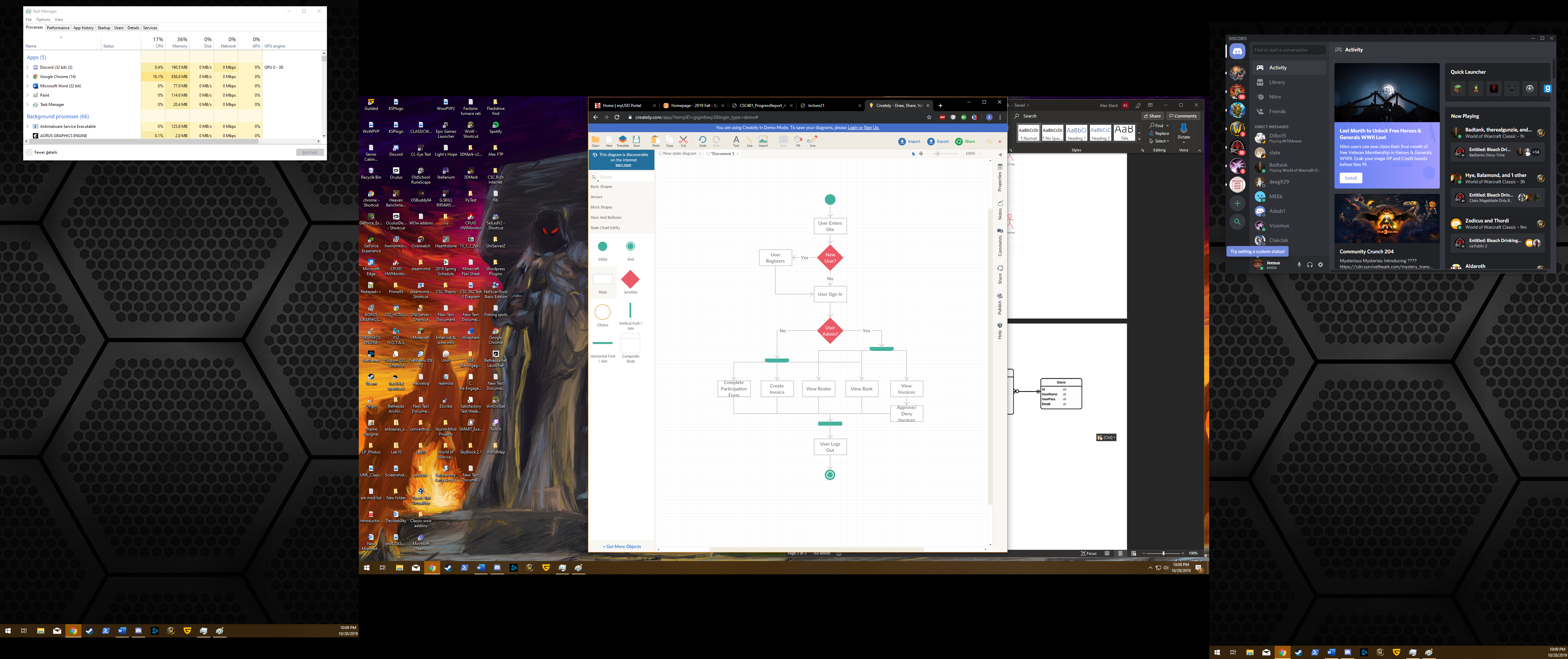
The site will require a differentiation between users and admins, with admins having different abilities than the base user. Because the admins of the guild are still doing the same actions as the regular members, they will have all of the base functionality granted to them of a user plus additional admin privelages. Admins will have the ability to review invoices and approve or deny them. Admins will also have the ability to add or modify items into the bank; moving much of the tedious work off to them instead of requiring the site admin to manually add them in when needed.



**Data design and database implementation:**

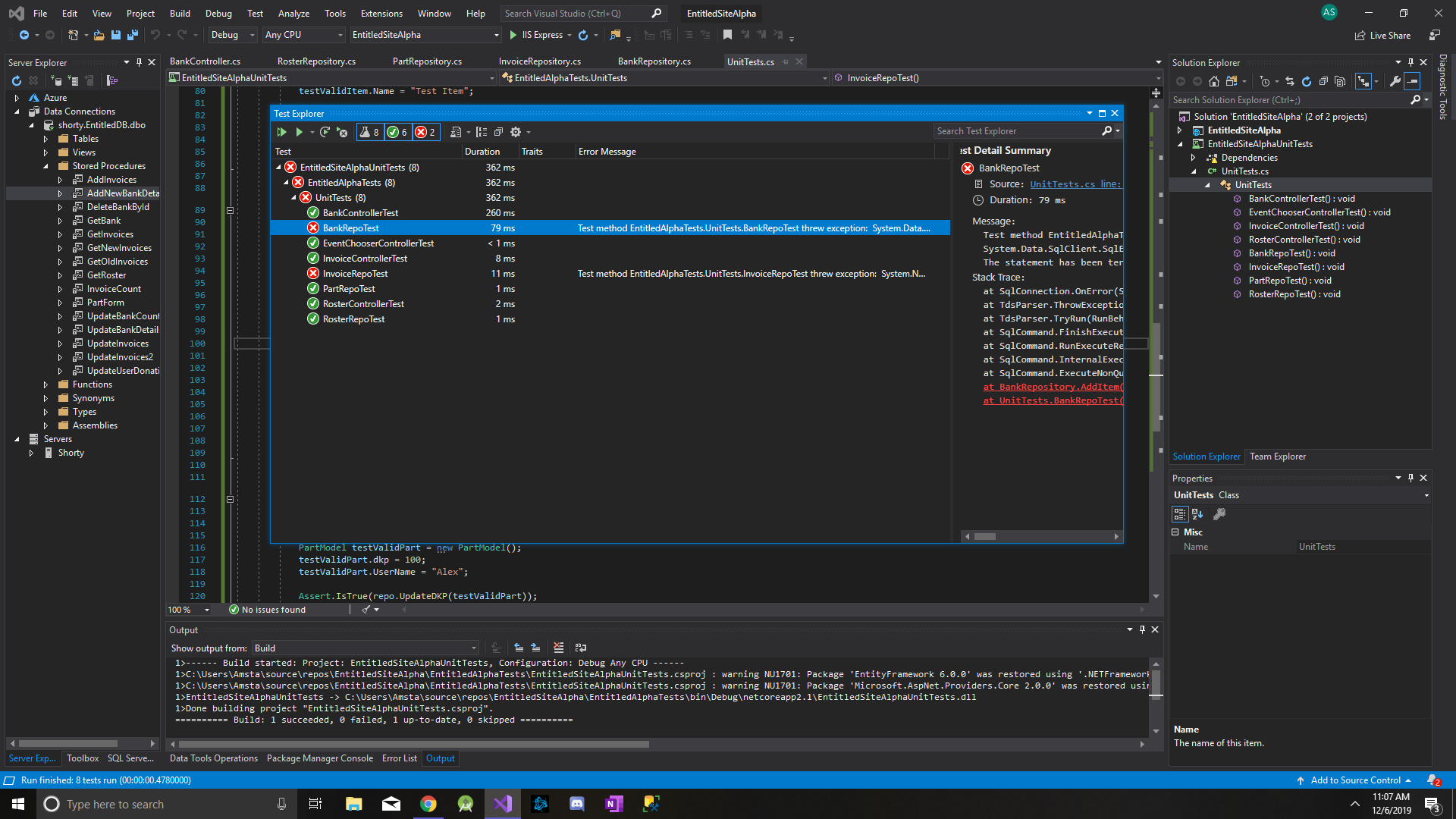
As shown below, the database will primarily consist of three tables: Users, Bank, and Invoices. The users table will keep track of the users of the site, with their included username, email, and their current DKP (participation points) earned. The Invoices table will contain all the invoices, current and past. Invoices are generated for one specific item, with the user having the ability to do a multi-item invoice and the site generating multiple invoices, one for each item. The bank keeps track of all the items currently in the guild’s possession, with the invoice system matching the request to the bank DB in order to make sure there are enough items to accommodate the request.

**Project Activity Diagram:**

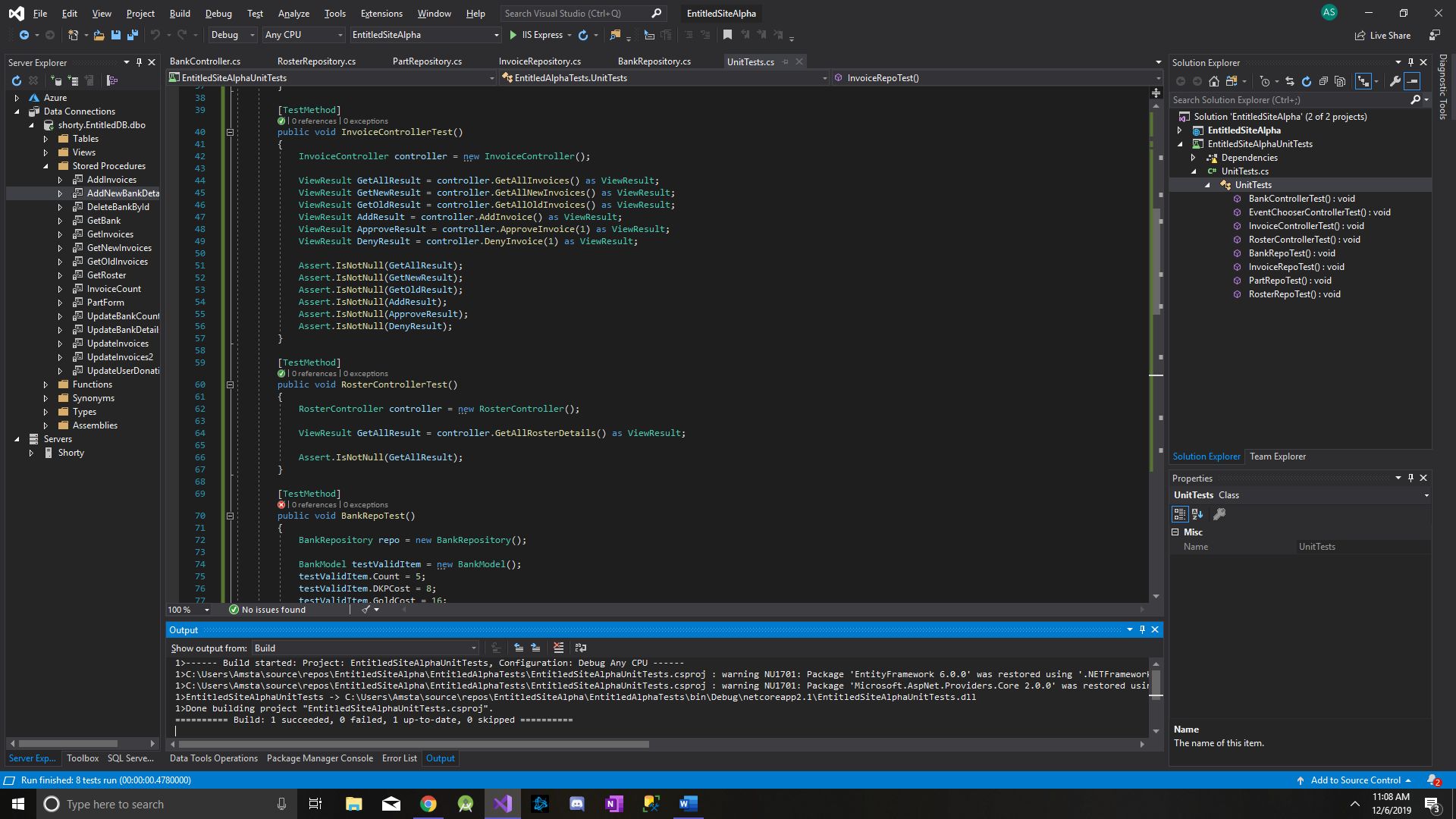


**Unit Testing:** Listed below are a few of the unit tests created for the project (Fig. 2), with a final report displaying the results (Fig. 1). The framework for doing the tests is included with .NET, MSTest. The tests are written in C# and contain a large sequence of tests which cover all Controllers and Repositories. The Controllers are primarily responsible for passing objects between Views(pages) and contain logic which either operates solely or calls the Repository functions. The Repository functions contain most of the logic and all of the SQL queries. The repositories are tested the most thoroughly, as that is where the majority of the C# code resides. The code does an excellent job of filtering the input, ensuring that only correct data is passed to the SQL queries.

**Fig. 1 Results of Unit Tests**



**Fig 2. A few of the Unit tests, displaying their format**



**Boundary-value analysis:** Below is the boundary-value analysis for the project. The project primarily uses two datatypes, integers and strings. Integers are used because the values being used will never have floating points, allowing the site to save space in that regard. The absolute maximum for the integers will always be MAX\_INT, and will never dip below zero (users cannot have negative DKP and the bank cannot contain negative items).



**Integration Testing:** With the way that .NET handles the passing of data, the integration tests are very important. .NET contains it’s data within Models, and passes those models between the Views (pages). Thus it is vital to ensure that the correct data is passed between modules to ensure proper functionality of the site, hence the importance of integration testing. Below are the test cases used to ensure that all pages which pass data are properly communicating. The modules contain data filtering, a robust feature included in the .NET package. This ensures that the user will be stopped before any incorrect data is passed between modules.

1. Bank Page – create new item

|  |  |
| --- | --- |
| Purpose | Ensure that admins can add new items into the guild bank |
| External Dependencies | Site must have valid connection to SQL database |
| Test Description | 1. Navigate to Bank page as admin 2. Click “Create new” to bring up create screen 3. Input valid responses (site does type checking) and click create 4. Admin is automatically redirected to Bank page, new item displays |
| Expected Results | The new item will appear in the database with the fields the admin had given |

1. Bank Page – edit existing item

|  |  |
| --- | --- |
| Purpose | Ensures admin can hard-code changes to existing items; admins may need to add corrections |
| External Dependencies | Site must have valid connection to SQL database  Given item must already exist in Bank |
| Test Description | 1. Navigate to Bank page as admin 2. Click “Edit” next to chosen item 3. Alter the necessary fields, click enter button 4. Admin is redirected back to page, given item has fields updated |
| Expected Results | The given item will have it’s fields altered with the values denoted by the admin |

1. Home page – create new account with valid inputs

|  |  |
| --- | --- |
| Purpose | New users must be able to register themselves with email address, username, and password |
| External Dependencies | Site must have valid connection to SQL database |
| Test Description | 1. User navigates to either Login or Register page (if login, user clicks “register” link) 2. User enters their new account info 3. User clicks create and is automatically logged in |
| Expected Results | New user is created in database with all DKP, Donations, and Invoices set to 0 |

1. Home page – create new account with invalid inputs

|  |  |
| --- | --- |
| Purpose | To test if the form validation control is working as intended |
| External Dependencies | Site must have valid connection to SQL database |
| Test Description | 1. User navigates to either Login or Register page (if login, user clicks “register” link) 2. User enters account info with either username, email, or password as invalid (such as email address with no @) 3. User clicks create |
| Expected Results | New account is not created and the user stays at the Register screen with error marks on the invalid responses |

1. My Account page – complete participation form

|  |  |
| --- | --- |
| Purpose | Users must be able to complete the participation form to get credit for guild events |
| External Dependencies | Site must have valid connection to SQL database |
| Test Description | 1. User navigates to My Account page 2. User clicks on Participation Form, is redirected 3. User checks all events done that week 4. User clicks enter, is redirected back to My Account |
| Expected Results | Given user is credited DKP for the events done. Each event is worth 8 DKP, so user must be awarded 8n DKP where n is the number of events done that week |

1. My Account Page – admin approve an invoice

|  |  |
| --- | --- |
| Purpose | Admins must have the ability to approve new invoices; approval must properly credit user and update bank |
| External Dependencies | Site must have valid connection to SQL database |
| Test Description | 1. Admin navigates to My Account page 2. Admin clicks View all new Invoices 3. List of all new invoices are displayed, with an “approve” and “deny” link next to them 4. Admin clicks “approve”, redirected to approval page 5. Admin can make changes to given invoice if necessary, clicks “approve” 6. Admin is redirected back to New Invoices page |
| Expected Results | Upon approval, the invoice must be marked as reviewed, removed from New Invoices, and sent to Archived Invoices. Approval must credit/debit the bank and DKP of the User who submitted it. |

1. My Account Page – admin deny an invoice

|  |  |
| --- | --- |
| Purpose | Admins must have the ability to deny and invoice |
| External Dependencies | Site must have valid connection to SQL database |
| Test Description | 1. Admin navigates to My Account page 2. Admin clicks View all new Invoices 3. List of all new invoices are displayed, with an “approve” or “deny” link next to them 4. Admin clicks “deny”, redirected to deny page 5. Admin clicks “deny” on bottom of page after reviewing details, redirected back to New Invoices page |
| Expected Results | Denied invoices do not credit/debit the User and bank and is removed from New Invoices, sent to Archived Invoices |

1. My Account Page – admin view all archived invoices

|  |  |
| --- | --- |
| Purpose | Admins must be able to see all previous invoices |
| External Dependencies | Site must have valid connection to SQL database |
| Test Description | 1. Admin navigates to My Account page 2. Admin clicks View Archived Invoices 3. Page populates with all old invoices |
| Expected Results | The page must populate with all previous invoices |

1. My Account Page – request item from bank

|  |  |
| --- | --- |
| Purpose | Users must be able to request items from the bank, generating a new invoice |
| External Dependencies | Site must have valid connection to SQL database |
| Test Description | 1. User navigates to My Account page 2. User clicks on Request/Donate link 3. Redirects to Request/Donate page 4. User selects Request, selects the item from a dropdown, and types in the number of items 5. User clicks Create, is redirected back to My Account page |
| Expected Results | Upon create, a new invoice is generated, with the request coming from that user. Admins can see new invoice from the View New Invoices page. Bank is not updated until admin reviews |

1. My Account Page – donate items to bank

|  |  |
| --- | --- |
| Purpose | Users must have the ability to donate items to the guild bank |
| External Dependencies | Site must have valid connection to SQL database |
| Test Description | 1. User navigates to My Account page 2. User clicks on Request/Donate link 3. Redirects to Request/Donate page 4. User selects Donate, selects item from dropdown, and types number of items 5. User clicks Create, is redirected back to My Account page |
| Expected Results | Upon create, new invoice is generated. Admins can see new invoice from View New Invoices page. Bank is not updated until admin reviews. |

**Future Work:** The website is intended for real-world use by my guild. I have been in communication with guild officials, where they stated a explicit need for such a product to simplify their jobs. The project will require further work on authentication, with a differentiation between Users and Admins being currently vacant from the site. Further polish is also required, cleaning and refactoring code as needed. I would like to bring on a front-end developer to better design the looks of the site, as my skills in web design are lacking. An additional feature has been proposed where integration with an in-game app, MonolithDKP, will be done. This will entirely eliminate the need for the participation sheet, as MonolithDKP takes care of that in-game. I have discovered that MonolithDKP has an API which my C# code can communicate with; the necessary code only needs to be implemented on my site.

**Conclusion:** In the end, I was very excited to take on this project. .NET programming is becoming ever-more common in the industry, vastly growing in popularity as time progresses. At this time, the most prevalent job title in the software development industry is for .NET programming with SQL databases. Because of this, I wanted to gain experience working with it to give me a competitive advantage in my budding career. I would highly suggest future students take on .NET projects if possible, as I learned valuable knowledge that I can take forth with me. I would also highly advise scholars to take on the subject in their courses, as there is a significant lack of such in academia as it stands.

**End Notes:** The site was made using the ASP.NET MVC 5 framework, coded in Visual Studio as a visual studio solution. I have chosen the ASP.NET core as the framework, as contrasted to the alternative Razer Pages. My database is a MicrosoftSQL database (MSSQL), hosting on my machine (LOCALHOST), with connection being done through connection strings within the Repository classes. Authentication is handled through integrated .NET Auth. The README.txt will contain explicit details on how to recreate the database, with included SQL commands to do such.

**References:**

Microsoft official tutorials on SQL connection - <https://docs.microsoft.com/en-us/sql/ssms/tutorials/connect-query-sql-server?view=sql-server-2017>

Bootstrap CSS themes used - <https://bootswatch.com/>

Tutorial describing in detail CRUD - <https://www.c-sharpcorner.com/UploadFile/0c1bb2/insertupdatedelete-in-Asp-Net-mvc-5-using-ado-net/>

Microsoft official tutorials on authentication - <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-3.1&tabs=visual-studio>

Major inspiration for site look and feel - <https://www.method.gg/>