BTech 450 Final Report

Dan Kane

Digital Asset Management System

Digital Stream

Abstract. The need for a single location for storage of digital assets that can be made readily available from any where, has been the greatest push behind the number of different digital asset management (DAM) systems coming out in the previous few years. This paper covers the current stage of my project, in creating a DAM system. It will go over DAM systems in general, and what the specific type of DAM system is that is being created. The methodology chosen and various reasons for it, comparing and contrasting different approaches to development and implementation will all be explained, as well as what processes that have been gone through to get to the end stages of development.

CONTENTS

1 Introduction

Digital Asset Management Systems are not a new idea, having been around for a least 10years already, but the evolving state of the Internet and line speeds has meant that they are much improved on what they used to be. In simple terms they are web-based systems, with an underlying storage system, normally a database. The web base system is usually displayed to the user using a thin browser based client, however some approaches have used a software based client that the user installs on their home machine and this then connects to the web based system.

2 Digital Asset Management Systems

Digital asset management is the storage and organization of digital assets, usually owned by a particular company. Some companies have their own in-house management system that may be accessed to store and retrieve the digital assets of the company. However not all company have the technical infrastructure to set this up, and as such, the opportunity is there to produce a system that can store and organize digital assets of multiple companies at one location.

This involves the creation of a server where all the assets can be stored, which is usually stored at the company that owns the asset management system. Because the server is not located at the companies wanting to store assets, it means that the system should be web based to allow remote access. Since this system involves the storage of potentially very private and confidential data, the storage and access of assets needs to be secure, where downloads are only allowed by those with permissions.

There needs to be accounts to store assets and accounts to download, normally there will be only one account to store, and multiple accounts to download, but this varies among different implementations of the system.

So as a summary a digital asset management system involves the secure online storage of digital assets that can be accessed remotely from any computer with web access, but only buy those with accounts that have the right permissions.

2.1 Digital Assets

A digital asset is any sort of digital information that has been created by a company and contains information that the company wants to store. This could be documents, images, video or even program code and user testing results, just anything digital.

2.2 Brand Asset Management

For this project the digital asset management system a brand asset management system, which instead of dealing with all types of digital assets, it just handles digital assets containing brand data. This usually means images of brand logos, videos of

advertisements, and flash introductions to the starts of websites. For this type of system the owners of the accounts that upload and store assets are companies with the brand data, and the accounts that download the brand assets are normally advertisement designers and creators that use brand logos and such in their advertisements. Therefore the assets that are uploaded must be high quality images that can be used in large advertisements such as posters and billboards. This means that the size of the server storing the data must be very large to handle many customers storing large data files.
When the owner places their brand assets into the system they set permissions on whether the asset can be downloaded or not, this allows the owner to put assets on the site that can be downloaded only when the owner wishes. The other feature is that if the owner can use permissions to set who can download their assets, allowing them to only set permissions for the advertisers that do the advertisements for the owners.

2.3 Reasons behind Development

The idea behind this project comes from the customers of Digital Stream. Digital Stream mainly focuses on web site development for clients, and often a client requires further work done updating a site, requiring the reuse of the high quality original images and video that was used on the first site. However Digital Steam does not store these brand assets for them, and when they come back for more work on their site they normally require these images for the new work. Most clients either assume that Digital Stream contains their assets for them, or don't think about them once their site is originally made. So when Digital Stream inquires to the location of these brand assets, the most often get replies of "Don't you have them?" or "I have no idea". This leads to clients asking if Digital Stream can store their assets for them, which is possible but requires large amounts of server disk space and some way of managing the assets at Digital Stream's end, they would much rather have the clients organize their own assets. This is the main reason behind the development of a digital asset management system for this project. The other is that clients can be changed monthly fees for the storage of their asset data, bringing in new constant monthly revenue to Digital Stream.

3 Project Development

When software is being produced, one of the biggest risks is producing the wrong software. The end product may be an amazing high quality application, but this will still fail if it doesn't meet the end user requirements, that is, doesn't let the end users complete the jobs the originally wanted to complete. [8] For this reason spending time fully researching and analyzing the problem is very important before jumping into the design and implementation.
My development of this project followed a mix of different methodologies that I adapted to the situation that I was in. I was the only developer on this project, working on all stages by myself and I had no end users to get into contact with, only my project sponsor at Digital Stream. My methodology began by following standard

methods but I only followed these for a couple of stages before splitting off into my own method, which is shown in figure 1. After the initial planning stages I began by finding out exactly what a digital asset management system was and what different options for asset management were already on the market. From the research results I analyzed what requirements I found to work out exactly what I needed to develop. These requirements lead to the initial designs of the system, and with those I could begin implementing.

3.1 Methodology

There are multiple ways of developing systems, such as iterative, waterfall, agile etc, that all encompass there good points and bad, and they all have their own situations where they should be used. For this project I have tried to choose a methodology that fits with a single person project development and fits with this type of project.

3.1.1 The Waterfall Methodology

The waterfall methodology is one in which systems are designed and developed in stages. The commonly used stages are Requirements specification, Design, Implementation, Integration, Testing and debugging, Installation, and finally Maintenance. The waterfall method states that each stage should be completed before moving to the next one. However this is where problems begin, as it is generally hard to perfect each stage before moving to the next one, and often when developing a system there is a lot of back and forth between stages. [1]

3.1.2 Agile Software Development

Agile methodologies develop software and systems by way of time boxes, which are short periods of time, typically around 4weeks that includes the development of a new functionality for the system. The time box contains all the processes needed to develop the new functionality, planning, requirements analysis, design, coding, testing, and documentation. Developing in this way, creating the system piece by piece (or functionality by functionality) means that risk is reduced. If the development is disbanded then the functionalities developed can still be reused in other developments. Agile methodologies use face-to-face communication rather than the production of documents. The development team (including customers) is usually located in one place where it is easy for communication. Instead of documentation at early stages, agile methodologies use working software to display progress. This results in a lack of documentation for these methodologies. [1]

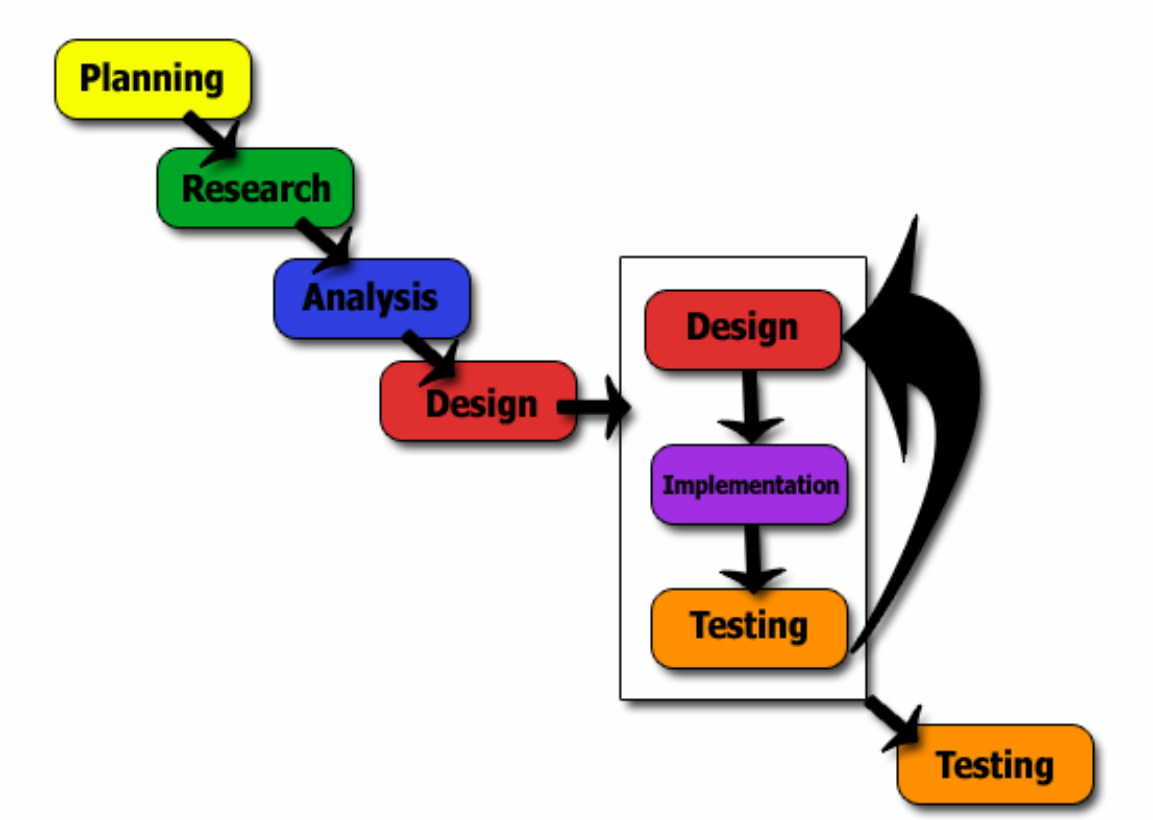

Figure1. Methodology Diagram

3.1.2 Project Methodology

The chosen methodology for this project ended up being a combination between waterfall and agile development. The initial stages were closely mapped to the waterfall methodology where each stage was completed before moving to the next. However when the first design stage was reached, after completing database and early overall system designs, the main tasks were split up into components and had separate design, implementation and testing phases. During each components development a prototype was designed, implemented and tested, then if it required further work, the prototype reentered the design phase and the process begin all over again. As the development of the system progressed, the time taken to develop a component was reduced as the results and experienced gained from the development of previous components could be used to speed up development. After all components are developed they need to be put together and tested as a whole system.

3.1 Planning Phase

The planning of this project was the first stage of development. During planning the main tasks of the development were identified and an estimated time of completion was made.

In this stage lack of experience in project development was made clear as the estimated length of time to complete phases was considerable off the actual time taken. However the fact that these were only estimates was taken into account and room was left incase any allowances needed to be made, this reduced the overall effect of wrong time estimates on the development.

3.2 Research Phase

The research process began by first looking at other companies that had attempted the same sort of project as the brand asset management system. Mostly New Zealand companies were looked at, but also a couple from overseas. New Zealand companies were more relevant to the requirements of this project, as they dealt with the same customer base as the system will have. For each system examined, a list was created containing the functionality provided, what made the system standout, and how could it be improved upon.

3.2.1 E-See

One of the systems researched into was E-see [2]. It was developed by a New Zealand based company that has been around since 1997. The E-See system is mainly used for brand data management, where companies and advertisers are the main clients. The companies can upload their logos and product images and advertisers can search for them and download. They divided their clients into two categories, Brand Owners and Brand Users. Brand Owners store their brand data by uploading it to the system and Brand Users download it. The system is free it use for Brand Users to download and search, but for Brand Owners they need to purchase a monthly plan for data space usage. They do have an option for a free Brand Owner account but this is only able to upload and store up to two items, so is only suitable for testing the system. This is one of the good features of E-See. Brand Owners also have the ability to restrict who can access their brand assets which is done via a permission lock on each asset. When uploading an asset the Brand Owner sets if the assets are available to general use or Brand Users must request permission before using it. So when Brand Users search for a particular asset, they will have to option to either download it directly or request permission first. Another one of E-See's strong points is that they do all picture processing for the Brand Owners. Any image uploaded is tidied up and then converted into multiple formats for the Brand User to download.

3.2.2 Original Image

Original Image is also a New Zealand company that created a Digital Asset Management System [3]. However they have developed it in a different direction than that of E-See, where E-See was directed at companies brand data, Original Image has directed theirs towards storing all of a companies digital assets in one convenient location. There is still the option to make the assets available to others, but there is no free user account option, permission is given out from the owner of the account, and they can define what restrictions are put on the account they give out. Original Image

also aims to store all sorts of digital assets, not just images, assets like video, audio and documents, etc are all supported.

3.2.3 Research Summary

Over all the system's looked at they had many parts in common, and only a few specific things each. The systems don't look to be overly complex, just a user interface connecting to a storage facility that gives the clients options to search, upload, download, and restrict. The research finding were taken into the analysis stage where the aim was to reduce all the systems finding into what Digital Asset Management System was and what functionality is should have.

3.3 Analysis Phase

The research phase had resulted in the list of potential functionality of the end system. The analysis phase dealt with using this list to refine it into what functionality the end system will actually have. Things such as time and resources had to be taken into account so that the system wasn't designed to contain too many functions and then never finished. This required identification of the definite functions of the system, things such as registration, uploading, searching etc that without would not be a useable system. Then the bonus functions were identified and sorted into groups of things that would be attempted to develop if time permitted. These were things like multiple views for search results, customized skins to change the look of the site and video previews. As well as identifying functionality, the analysis phase also dealt with identifying the users, or actors of the system. Once these were identified the actors were assigned functionalities of the system.

3.3.1 System Actors and Functions

Users
Users are those that signed up to download assets. These accounts would be free to signup to, and come with no monthly fees. Users would need to perform more tasks than a customer but less that an owner. The user needs to be able to login to the site using their ID and password. They need to be able to initiate a search for the asset they want, this will be spilt into two categories, simple and advanced. The simple search will be able to be accessed from the user's main page and just consist of a single text box input, which will search through asset names, companies, and descriptions. The advanced search will have its own page, and give the searcher multiple choices on what to search under. They will be able to specify what terms they which to search and what type of search they want to do. Search results will be displayed to the user and they will give the option to download, add to cart, or if the asset is locked, request permission. The add to basket option is similar to a shopping

basket in online stores, but instead of paying for multiple goods together you can download them in a compressed form together. The download sends the file to the user directly using HTTP. If the asset is locked and the user needs to request permission then they can request it by submitting a request reason message that will be sent to the owner of the asset. When the request had been granted of declined by the owner, and email will be sent to the owner informing them of the owners response. Users will also be alerted to any new responses when the first login to the system. Other user functions include being able to edit their accounts, and to signup as an owner.

Owners

The owner has many more functions than a user, so more time was required on analysis and design on their part of the system. Their main functionality adding an asset to the database needs to allow them to specify how their asset is described exactly, this was done by have preset fields for them to fill out as well as custom defined keywords that can create descriptions specific to a single or group of assets. The Owner will be able to add assets to the system either by sending a DVD or some other kind of media device in with the assets on it, or directly via HTTP upload. When they add an item they can also specify the permission status of the asset, if anyone can download – unlimited or limited time – or if they need to request permission first. If the send assets in via DVD then the administration at Digital Stream will add these to the server under pending assets for the owner. The owner can then fill out the description details and permissions of the assets to fully add them into the system. Owners need to respond to requests from users, and any requests that haven't been responded to will be shown to the owner when they login as well as sending them an email when the request was made. The response is either to decline or accept, with an optional response message to the user. Another functionality is being able to view and edit their previously added assets, which will encompass a search tool to let them search through the assets stored in the database, but restrict the search to the current owners assets. Here the search function from the user will be reused to save time. Creation of sub-owners is another one the owners' main functions, here the need to be able to choose sub-owner details and set their permissions. Other owner functions include editing their account, changing payment plan, making payments, managing sub-owners, and viewing reports.

Sub-Owners

The sub-owner has many of the same functions as the owner they are just limited in the tasks they can perform. For example an owner could create a sub-owner that could only respond to requests but not upload, edit or delete the owner's assets. This allows the owner to lessen their load, so if a marketing department of a company had an account, the marketing director could be the owner, and they could make multiple sub-owner accounts for the workers under them. Then the sub accounts could do the majority of the uploading and responding.

Administration
The administration will be employees of Digital Stream making updates to the assets in the database and the client's accounts. If an owner sends in assets on DVD then an administrator will need to add these new assets to the server under pending assets to the owners account. Administration will also need to be able to edit any assets in the database, including deleting them from the system, just incase anything illegal is uploaded. Owners who haven't paid their bills will have their accounts suspended until payment is received so administration will need to be able to do this as well. The other main functionality is report viewing, for this I would like the administration to be able to create their own sort of report, as well as having some commonly used reports preset in. Things like owners who haven't paid, database capacity, daily or monthly system usage etc.

3.3.2 Activity Diagram of Common Activity Flow

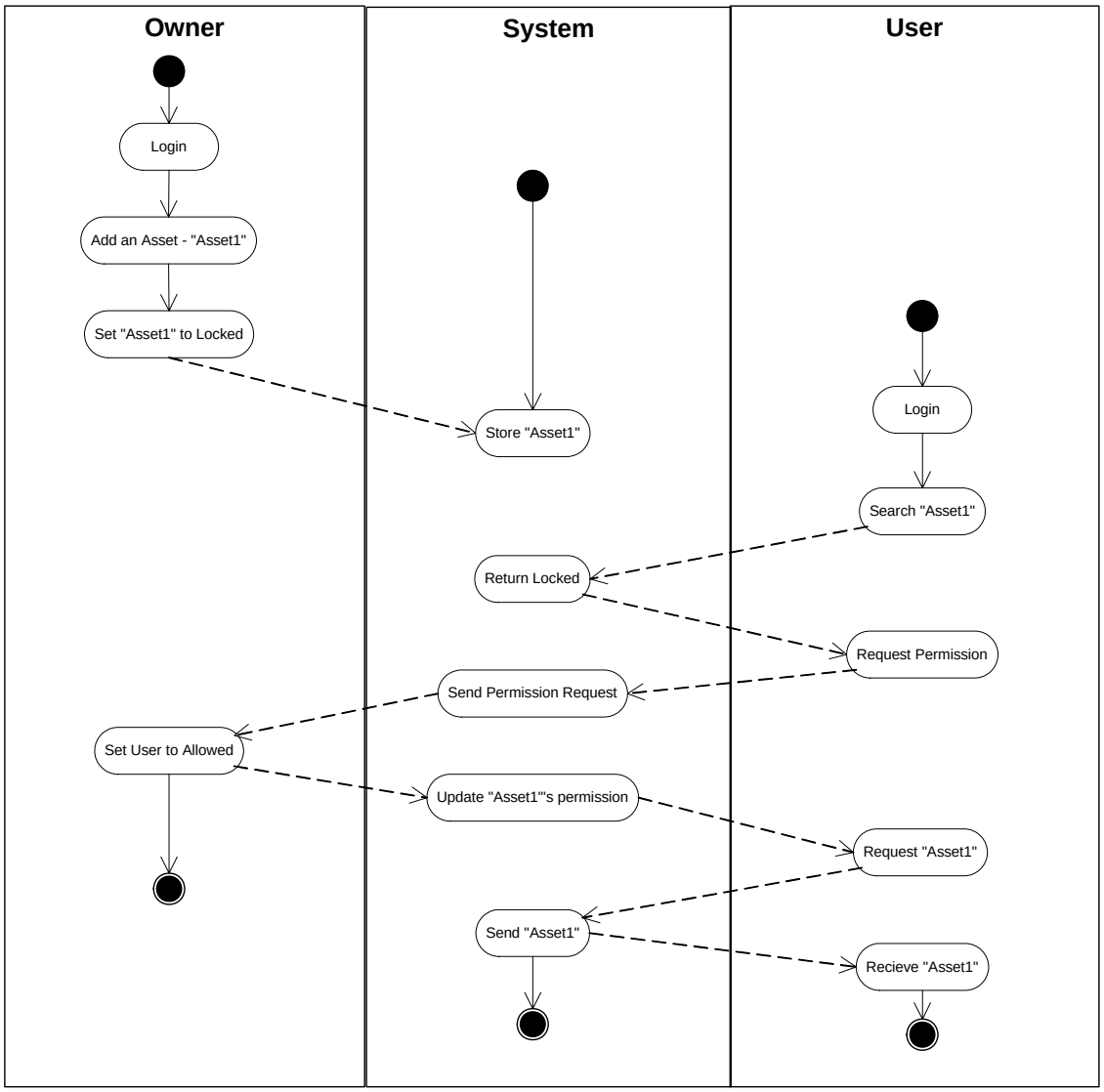

Figure2. Common Activity Flow

Figure2 shows the common flow of activity by actors interacting with the system. It begins with the owner logging into the system, and adding a locked asset to the server. A user then logs in and searches for that assets. The system returns that the asset is locked and the user must request permission to use it. The user then creates a request message and sends it to the system which sends it onto the owner. The owner reads the message and creates a permission allowing the user to use the asset, the system then updates the permission and when the user next requests the asset, sends it to them.

3.3.2 Use Case Diagrams

Use case diagrams graphically depict interactions between the system and actors, they show who will use the system and in what way the actors expects to interact with it. Use case modeling begins with identifying, defining and documenting actors that will use the system, and after this the use cases that the actors use also need to be identified, defined and documented. The final step is to identify any reuse possibilities where use cases are found that maybe have different outcomes and different actors interacting with them, but have some similar steps involved in the carrying out of the use case's task. Reuse identification means these common steps are removed from each use case and put into a single abstract use case that can be used by other use cases. [4]

An example of this is the owner managing their assets and a user downloading an asset. Both use cases involve the actors performing a search through assets stored in the system by entering a search term and then selecting an asset to perform a search upon. Even though the action performed by the owner is editing an asset and the action performed by the user is downloading an asset, the search part of the use case can be made into an abstract use case.

Use case diagrams are useful as they show the actors and their functions in a graphical form that can be easily understood, which was used to display the intended system to the project sponsor at Digital Stream. They also show how the functionalities are ordered and how they interact with each other.

The other side of use case diagrams, the documentation, is also very useful in explaining the flow of interaction between actors and system, in the descriptions of the functions that they need to perform. The documentation contains the steps that the actor associated with each use-case goes through, including normal flow, alternate flow, error flow. The steps for each use-case can be further reused throughout the development stages, in database design through to interface development.

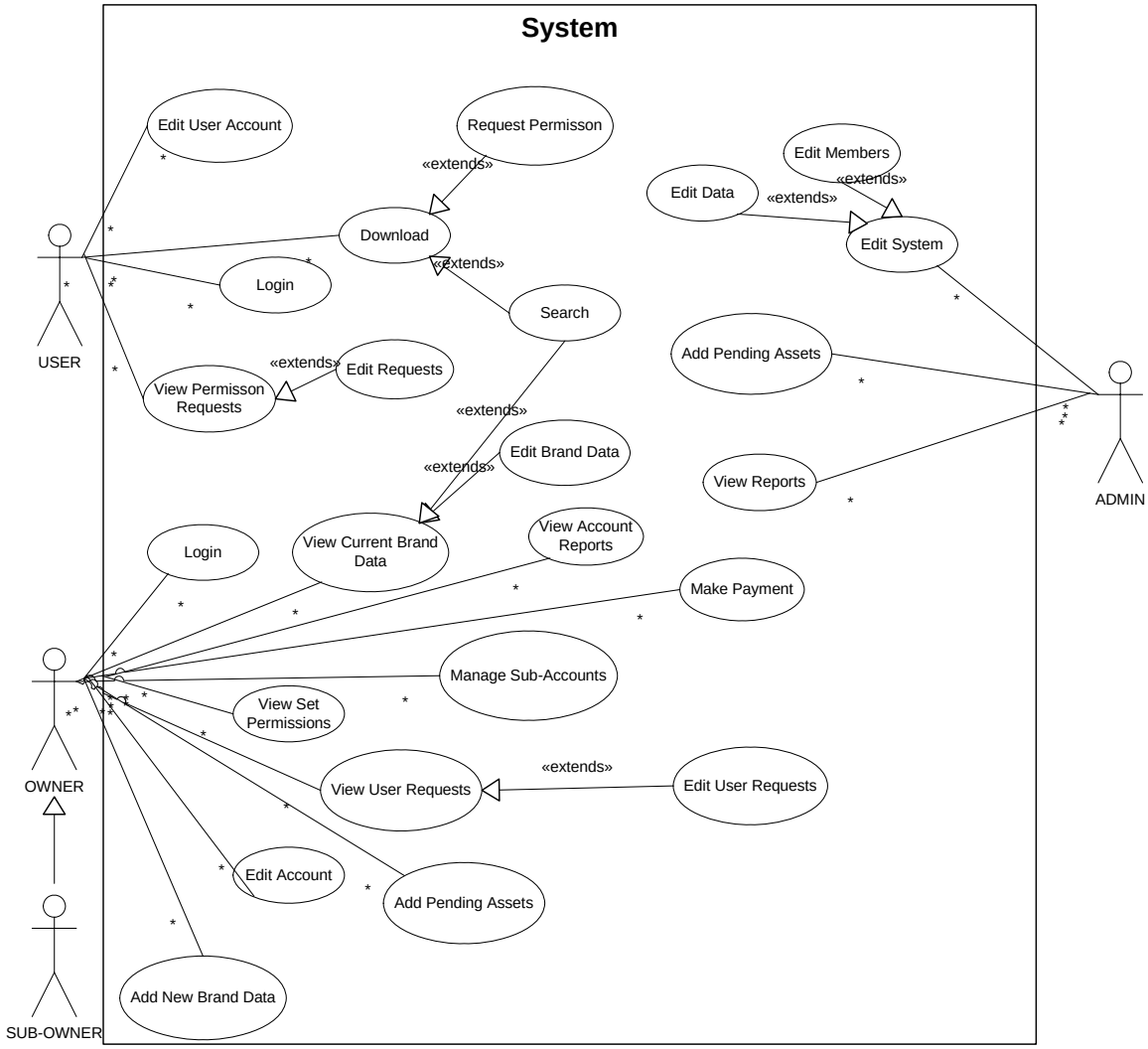

Figure3. Use Case Diagram

The use case diagram clearly shows the four actors of the system and the tasks they can carry out. As expected owner's tasks take the majority of the systems functionality, and reinforces the decision to spend the majority of the development time on their part of the system.

3.4 Design Phase

The design phase of the system development lifecycle is defined as those tasks that focus on the specification of a detailed computer based solution to the initial problem that the system will solve. Here the problem changes from being defined as from as business point of view where actors were identified and there tasks defined as in the research and analysis phases to now being focused on the technical side of things. [4]
When design is done, the majority of designers design for the whole system, developing designs for the system to be developed by themselves or their development team. They

design inputs, outputs, files, databases and other computer components, but in reality most companies will prefer to purchase more software then write it in house, [4] as it leads to fast development of the finished product, and you can normally – after research – be sure that the software you are purchasing will be working a bug free. Most purchased software will also allow be able to be modified to fully fulfill you end users requirements. For this project however, there was to be no purchasing or outsourcing of development so all parts of the system needed to be defined at this stage. Of course the other option is to use free open source software, and this option requires very extensive testing to make sure it is bug free before the systems release.
The design phase of a systems development is very important to the overall success of the system. The whole implementation phase is based on what is designed here, and a bad design means that the implementation will fail and the design will have to begin all over again. The design phase involved designing all three of the layers of the system, shown in figure4.

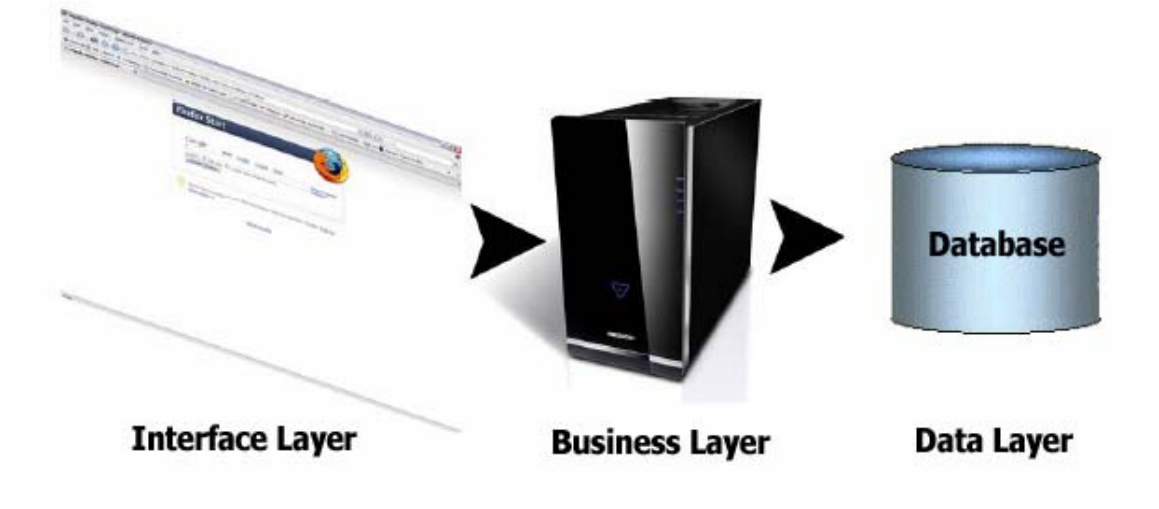

Figure4. Layer of the System

The interface layer was to be web based and developed in ASP.Net, the designs for this needed to deal with the layout of the pages, the navigational flow between pages, and the controls and decoration contained on them. The business layer was developed in C# and as such needed designs on classes and methods required and the hierarchical structure they had. The final layer of the system, the data layer was a SQL Server database, and here the design dealt with the tables required in the database, what fields they would require, and any procedures that would need to be created.
Design started with the interface, working out which pages were required, then what needed to be on each page. From here required tasks from the business layer were added at each page, and then any database connections that would be needed to be made.

3.4.1 Interface Layer Design

The interface layer began with the navigational page flow diagram, which represented which pages the interface would require and how each of these would connect to one another. This diagram is shown in figure5, and you can still see that the owner take precedence over the rest of the actors, with much more pages dedicated to their section of the site.

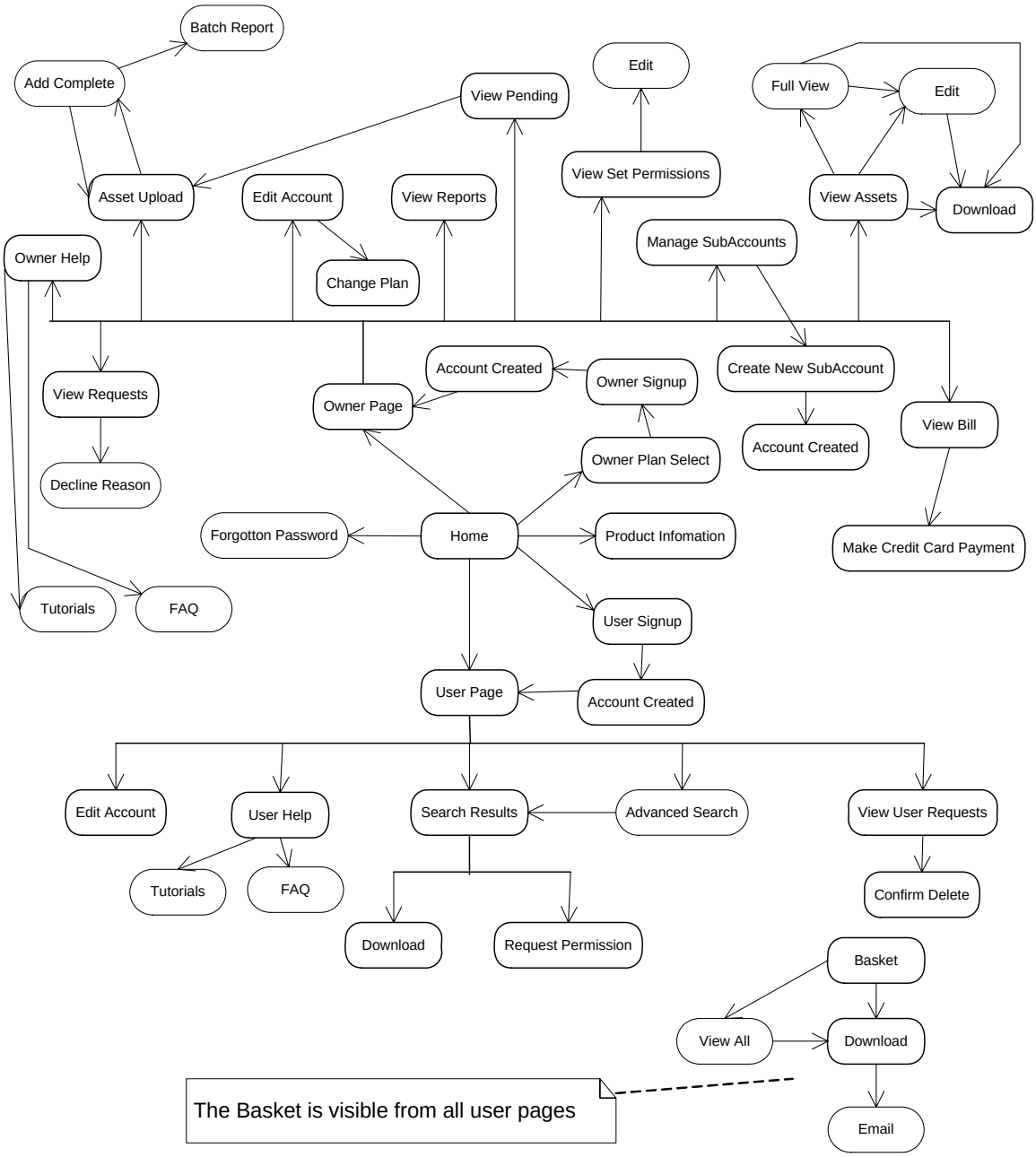

Figure5. Interface Navigational Flow Diagram

3.4.2 Overall System Design

From this navigation flow diagram the overall system diagram was created, this diagram was created in steps, with each step adding a little more information about the system and its tasks. It began similar to the navigation flow diagram but the controls required on each page were added and the flow was changed to flow from buttons not just pages. This resulted in a diagram similar to that in Figure6 but without the system tasks. From this the tasks required by the system were added and what controls would activate these tasks, finally the database connections required by the system tasks were added completing the overall system diagram, from which a sample has been taken and shown in Figure 6.

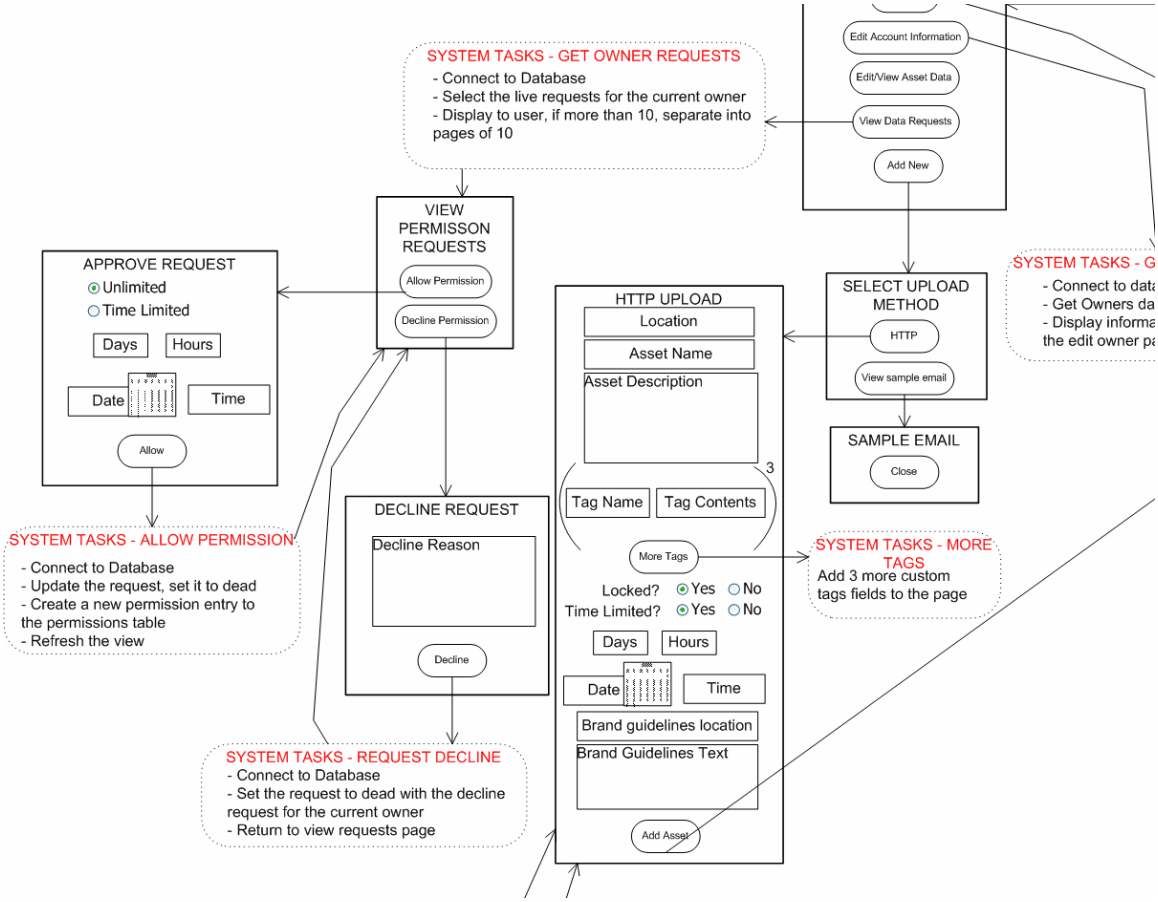

Figure6. Sample of the Overall System Diagram

3.4.3 Database Design

The end goals of database design are to have efficient storage, updates, and retrieval of data. The stored data needs to have high integrity to promote user trust in the data, the database should also be able to adapt to future system changes, and handle multiple requests from users and once. [4]
Database design is normally done is steps where logical and physical models are designed and improved upon in an iterative manner. The first step is to create a logical model that describes entities and their attributes. An example of an entity from this project would be an owner, and attributes would be like ID, first name, address etc. The logical model should also create entity relationships, connections between multiple entities. An owner entity would have zero or more assets entities, and exactly one owner type entity.
After this model is complete the next step is to perform a task called database normalization, where logical inconstancies and anomalies are removed. There are three different types of anomalies, update, insert and delete.
An update anomaly occurs when the same entity's attribute ends up with multiple versions of the attribute value on different tables. Like an owners address, if it was stored in more than one table and they wanted to change their address, if the change only happened in one of the tables an update anomaly would occur.
An insert anomaly occurs when a new record can not be inserted into the table at all, when it should really be able to. This can occur if you set up your entities such that they contain attributes that are not defined by the identifying attribute of that entity. If the owner entity contained sub-owner id and name attributes, and you wanted to insert an owner that at that point did not have a sub-owner defined, then the insert would fail and an insert anomaly would occur.
A delete anomaly occurs where a delete removes more information than you originally wanted from the database. If the asset entity contained attributes relating to owner details instead of an owner and asset being separate entities, and if an owner only had one asset, deleting this asset would remove all knowledge of that owner from the system.

Database normalization is a generally a three step process, where each step improving the design of the database in some way. The design first starts in its original form, where anything goes. It is just an initial design of mapping the business problem into a technical one. An example of this form is in Figure7.

Owner

ID	Name	SubOwnerID1	SubOwner Name1	SubOwnerID1	SubOwner Name1	SubOwnerID1	SubOwner Name1
o001	John Smith	s0001	Joe Cole	s0002	Conrad Smith	s0003	Chris Cains
o001	John Smith	s0004	Tana Umanga	s0005	David Smith	s0006	Lance Cains
o003	Kevin Johnston	s0007	Helen Clark	s0008	Dan Carter	s0009	Craig Leyhy
o004	Jing Qin	s0010	Carl Haymond	s0011	Locky Elsion	s0012	Marget Heminway

Figure7. Owner Table in Original Form

The owner table contains repetition of the sub owner id and the sub owner name. In this case John Smith has more than 3 sub owners so he has two rows in for his data. This can cause update and delete anomalies later as his information is spread over more than one location. Normalization fixes these problems, and stops the anomalies from occurring.
First normal form or 1NF is where the table is fixed to remove all nullable columns and repeating rows. This is done by taking the enforcing that each row must have a distinct primary key that identifies it completely. This will normally be ID and as such the second row of John Smith must be removed, as both John Smith rows have the same primary key.
Second Normal form or 2NF continues on from 1NF and defines further restrictions on what field can exist in a table. 2NF requires that each attribute in a table is only dependant on part of the primary key of that table. If they are not then the non dependant fields must be removed and put into their own table.
The final form is third normal form, or 3NF. Here the restriction is that every attribute that does not make up part of the primary key of the table must be non-transitively dependent on every primary key. The owners table contains the attributes sub owner ID and sub owner name. Sub owner ID depends only on the owner id as the owner sets the sub owner id, but the sub owner name depends on the sub owner id not the owner's id. For this reason the tables are spit resulting in the tables shown in Figure 8.

Owner

ID	Name
o001	John Smith
o001	John Smith
o003	Kevin Johnston
o004	Jing Qin

Sub-Owner

SubOwnerID1	SubOwner Name1
s0001	Joe Cole
s0004	Tana Umanga
s0007	Helen Clark
s0010	Carl Haymond

Figure8. Tables in Third Normal Form

The next step in my design was to construct the designs for the tables in the data layer. Having already designed the overall system made this job much simpler, the required controls for each page almost mapped one to one onto database tables. For example the asset upload page of the site contains controls for asset name, description, locked, time limited, etc and the asset table in the database contains fields for the same information. Of course not all controls map exactly onto the database, having 3 keyword tags stored in the asset table would violate the normalization rules [1] so instead keywords get their own table and a row from the asset table contains zero or more keyword table rows. Doing this reduces the amount of space taken up by the database table, as there would always be three fields in the table taking up space even if the owner didn't enter any keywords when they were uploading an asset. The final design for the database is shown in figure7.

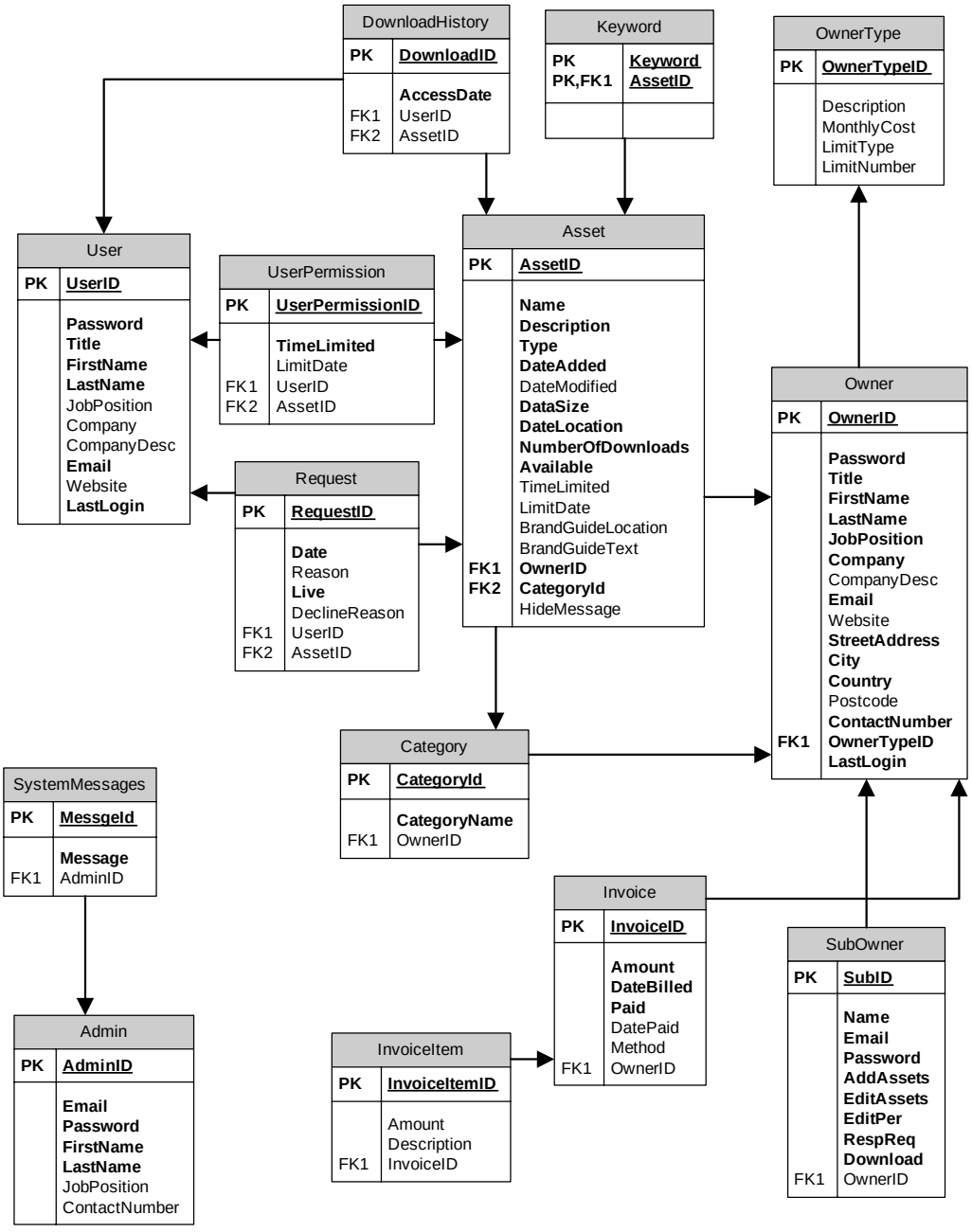

Figure9. Database Design

Some of the design decisions made here were to include a download history table, the reasoning behind this was to allow owners to find the users who were most often downloading their assets, and then have a function that allowed them to set it automatically that that user could have permission to their locked assets. This was assuming that the users downloading the owner's assets most often were the ones the owner wanted to download.

The owner table actually stores the amount of assets that have been uploaded and the total spaced used storing them. This is redundant information and doesn't follow the normalization rules as it can be calculated by counting the assets belonging to that owner

and summing the size of the data files, therefore not being dependant on the owner's primary key but on the assets in the asset table instead. However including these fields in the owner table vastly reduces the amount of time spent when requesting this information, and as this information is often requested, this helps optimize the system, at only a small cost of storage space. This same reasoning is behind the number of downloads field in the asset table, this information could be calculated from the download history table but with a much slower response time. The number of downloads is a commonly requested asset field, as owners will often wish to view their assets ordered by number of downloads to find out their most popular assets.
The practice of ignoring some of the normalization rules is called de-normalization and is an essential part of optimizing a database to make it efficient to use for systems data storage.
Although the administration table and the system messages table do not interact with the rest of the tables in the database, they are still vital to the overall system design. The administration table stores the id and password of administration to stop those with out authorization accessing the administration part, and the messages is for storing any sort of messages the administrator wants to broadcast to the users and owners of the system. Messages like system downtime or new account options would be stored here, and accessed when users and owners log in to the site, then if they have not yet seen the message, then it can be displayed to them.

3.4.4 Component Identification

After the overall design for the system was developed, including interface, business layer tasks and database, the next stage was to break up the system into components that could be designed piece by piece. Having components allowed me to design, implement, and test the system in parts, which increased development speed, allowed reuse of code, and allow the development of the system in the same flow as how it would be used. For example I started by developing the login component, then the part of the owners section where they upload and manage assets, before jumping to the users section, developing the search, download are request components. The added bonus of this meant I had test data pre made from previous components in the database already, so I didn't need to make any up.
Components were generally identified by the user interface pages of the site, but some pages would contain multiple components and other components were spread across several pages. Each component contained a graphical user interface, business layer code and normally some database interaction. The components identified and the order they were developed are contained in list1.

Components of the System

1. Registration – User/Owner
2. Login – User/Owner
3. Authentication – User/Owner
4. Main Menu – Owner
5. Top Header – Owner
6. Asset Upload – Owner
7. Asset Management – Owner
    a. Editing
    b. Deleting
    c. Downloading
8. Main Menu – User
9. Top Header – User
10. Asset Search – User
    a. Simple
    b. Advanced
11. Asset Download – User
    a. Unlocked Assets
12. Cart – User
13. Asset Permission Request – User
14. Request Management – User
15. Handling Requests – Owner
16. Permission Management – Owner
17. Asset Download – User
    a. Locked Assets
18. Messaging – User/Owner
19. Account Management – User/Owner
20. Billing Management – Owner

List1. Component of the System

3.5 Component Development

For each component there needed to be a user interface designed. This was generally a single web page but for some components it only meant designing multiple pages, like for the cart, a small design was required to show the cart on each page, but the page where the owner would view and edit the cart was also required.
As well as the interfaces the business layer needed to be designed, this included any classes that would be required and what methods would be used.

For the database interaction, the tables that needed to be accessed and how they would accessed needed to be decided and designed.
Following the completion of a components design, the implementation was started. The implementation began from the lowest layer, the data access. So before anything else was done all database connections and SQL commands were created and testing. Testing involved giving sample data into the methods and making sure they all did what they were supposed to.
The second stage of implementation was the creation of the components business layer. Classes were created and the methods that were required were written. Most methods involved taking some inputs, which would come from the user interface layer, and then performing some checking, and modification, before sending to the data access layer to store in the database.
The final stage, and the most time consuming, was the development of the user interface. This was written is ASP.Net 2.0 which I had never used before which meant I needed to spend a large portion of time learning how to build each page beginning. The pages were developed and the post backs to the server from the clients actions called the methods and created the classes that were developed in the business layer.

3.5.1 Interface Design Theory

There is a huge amount of theory out there describing how a graphical user interface should be developed and what things you should and shouldn't do to your end users. A lot of the information is common sense but there are good sets of principles and ideas using the psychology of humans and computers, and how they interact with each other.

The Psychology of Humans and Computers

Because of the limitations in current voice recognition and electronic response, user interaction with computers is largely visual based. Research into human sensors and perceptions is important to take into consideration when developing software. If a message is displayed to a user then, it needs to stay displayed long enough for the user to read and process the information before disappearing, its no use showing messages to users if they don't have time to read what they are. Simple basic human limitations like this need to be understood to determine the time period that a message stay displayed to the user. Humans are also easily distracted by changes in our perceptual system. This can be changes in sound, color, movement, or contrast, all of which can distract from the current task. This is why changes in the system should only occur to keep the user more focused upon their current task, or alert them if an important action is waiting for them to act upon. The university system does this quite well, if the user hasn't been active in their mail or Cecil session for a certain period of time, they are alerted that if the don't make an action they will be logged out, but otherwise nothing happens to distract the user from what they are currently doing.

Humans have a thing called sensory storage [5] which is where the automatic processing of sensors is taking place. Human sensors are constantly alert, taking in your surrounding

and processing them. Applying this to user interface design means that having a pretty animated background to your site may look nice to the user, but when they are performing tasks there sensors are constantly using your sensory storage to process the movement in the background and discard it, as not being worth your attention. Using the storage like this leaves less for the actual processing and storage of the useful information that relates to the task at hand.

Humans have two types of memory, short term and long term, both of which need to be taken into account when developing a user interface. Short term memory is normally limited to between five and nine items and can only be maintained for a short period of time, with most people having a maximum of about 30 seconds. [5] To deal with the limits of short term memory people usually use external memory aids, such as writing something down on paper. For computing the external memory aid can be text on screen, either given by the server so the user doesn't need to write anything down or a control field where they can enter their own text. The problem with that is that pages are not constant and can be changed completely or have other pages popped up over the top of them. This is especially relevant when displaying help to users, interfaces need to be designed so that the help page doesn't cover or replace the information the user was trying to get information on, often when you look up the help for clarification on a certain feature, if your taking a large amount of time to find the answers, often you can forget what you were looking for and need to refer back, of course if your help is replacing the information you wanted help on, you cant refer back to it easily.

To use long term memory in computing means reusing names, key shortcuts, and icons that are commonly used in other systems, like a minimize button always looks the same and is generally located in the same place, if you put a minimize button in your system that looks different and is located differently you are not taking advantage of users long term memory and forcing them to learn something else in place of what they already know. Retrieving long term memory occurs by recognition and recall, ask a causal computer user what the maximize button looks like and they probably can't answer, but show them an image of the minimize, maximize and close buttons together and ask them what the middle button is, and they will be able to tell you it's the maximize button.

Another feature of humans is that we make mistakes, we always have and we always will. There is nothing that can be done about this except to make systems as easy to use as possible. Any feature that may be a little confusing needs clarification on use and intended results. Even the best system will still have users that make mistakes, so a system design must take into account error handling and a back function. One of the best functions of computer systems is to be able to undo what you just did, accidentally delete the wrong file, select undo, and its back. The other option to reducing user errors is to use confirmation boxes on all important functions, if a user clicks delete, ask them if there sure they want this file deleted. Of course to advanced computer users confirmation boxes can be very annoying, so they should be used only in the most important system altering situations that cannot be undone. A good example of overuse of confirmation boxes is EA Games Rugby 08, where you need to confirm almost every action you do, "Are you sure you want to change to this setting?", "Do you want to save these new

settings?", "Are your sure this is the right settings filename that you want to save?" , and finally the most annoying confirmation box of all "Are you sure you want to exit?" If you are developing a system with this many confirmation boxes then there must be an option for not showing this confirmation box again.

User Design Principles

In the past systems were developed without much thought to the end user, meaning that users had to adapt to the systems, whereas these days the systems are developed very much with the end user in mind, and the end system should adapt to the user. Design principles are not GUI design rules, they are based on the cumulative wisdom of many people, compiled over several decades of experience in designed interactive systems for users. The main principle that all other principles can be said to be derived from is "Focus on the user not the technology" [6] this means that users are the most important factor in the design of the system, the interface shouldn't be designed to display the feature of a wonderful piece of coding, but to allow a user to complete all their required functions. It doesn't matter if the developer thinks their page looks fantastic and it simple to use, it's the end users opinion that matters here.

#1 Place Users in Control

Putting the user in control means letting them decide what tasks they are going to perform, how they will perform them, and let them quit when they want. Users should be able to choose how they put there inputs into the system, allowing the user of both the keyboard and mouse is very important, not only for giving the users freedom but also for accommodating different levels of users. An inexperienced user filling in a form will often use the mouse to move from control to control and click on then submit button when they are finished, but an experience user will just use the keyboard almost exclusively and just tab their way through the form and submit it with a simple press of the enter key. A form must be able to handle both and make sure that when tab is pressed the focus jumps the control users would expect.

Messages and labels displayed to users need to be descriptive, if there is an error in a form submission the error message "Bad form input" is quite useless to most users, but if the error message is "Incorrect input for Email" and the email control is highlighted then the user can know exactly where the problem is.

Users also need feedback on what's happening in the system, if something going to take a while then tell them, and if possible have some estimate on how long the task will take. When they move over a control they can click on, give them some visual feedback that they can click on this control.

Navigation through the site must be easy and follow common sense paths, if there are back and next buttons, it should make sense to the user what the page that appears on the next click is. There also should be some context to show the user where they are in the interface structure and how they can get back to the home page. Exiting a logging out functions should also be made available at all times.

#2 Reduce Users Memory Load

The limitations of humans have already been discussed above, and the user interface design should take advantage of what the human can do and provide for places where there are limitations. Short term memory should be reinforced by displaying and recording the information they are working with, not forcing the user to remember this information themselves. A user shouldn't need to remember and repeat what a computer could store and then display back for them. Examples of bad systems in this area are dealing with phone agents, that ask you for information, and then a few screens later, ask for the same information again. A user should never have to input the same data twice. Long term memory should be enhanced using recognition to reduce the users memory load, instead of having them need to enter text into a field, if the option is there, make the control a list instead, where they can recognize and select an item rather than needing to remember and write it down. [5]

There should be visual clues giving the information on where they are what they're doing and where and what they can do next. This can be titles telling the user which page of the site they're in, and labels describing what control they're editing at the moment.

To help the users understand the system, and remember what things do, real world metaphors should be used to describe and visuals components of the system. Common examples of these are if you have a system that can make computerized phone calls, make it look like a phone, this lets users become instantly familiar and comfortable with using the system. The problem with this is if your component doesn't exactly map to a real world object, but you still use that object as a metaphor, it can confuse the user more than if you had created something new.

The final part of reducing memory load is to promote visual clarity, which can be done by having groupings of common controls in a table, either by using layout or color. The amount of information presented to the user should also be the minimum you can display while still getting you message across. Instead of having all the text on one page break it up into groups on separate pages and just provide links to the groups on the start page.

#3 Make the Interface Consistent

Consistency is a key aspect of usable interfaces, it can hugely reducing learning times, and reduce user error. However it is also a major area of debate, [5] consistency principles should only be applied if it makes sense for a system to follow the principles. Interface consistency can be consistency across common systems users use, or consistency inside a single system. The majority of users will be comfortable in the windows environment, so reusing icons and terms from there can provide consistency that makes users feel that they are immediately familiar with the system, even if they have never used it before. The system can and probably should be consistent across all of its pages, if a button does something on one page, then a button that looks the same on another page should perform the same function. Having page color and layout similar across a systems pages also allows a user to more easily find information and be more comfortable when the reach a new page for the first time.

3.5.2 Site Security

As this is a web based system, that can be storing private and confidential information, security is an important factor to take into consideration. There are three main security threats to take into account. The first is cross site scripting, which is where the user puts web scripting into an input field, and when the site displays this information later, the scripting is run. This could be something simple like an alert box saying “hello!” but also they could put some scripting which does something that you defiantly don’t want happening. This threat can be removed by checking the user inputs for any scripting text, and either removing it or using quotes so it is only displayed as text.
The second type of security treat is SQL injection, where the inputs that are taken from user and then used in SQL commands. The inputs could potentially change the function of the SQL command from a harmless select statement into a delete statement the removes all tables from the database. To remove the threat from the system the SQL commands can be produced with parameters in C#, which remove all characters that could change the commands function.
The final threat comes from the fact the owners are able to upload files that are stored on the server. If they uploaded a file with the name of a file already stored on the server, then that file would be over-written, this could potentially lead to huge problems if the named an aspx file something like “../Home.aspx” which redirected users to a different site. This threat is removed completely from the system by renaming all files that the owner uploads to one set by the server, and stored in a folder set by the server.

3.5.3 Registration Component

Data Access Layer

The tables accessed by the data access layer for registration are OwnerType, Owner, and User. Owner type is access to select the different owner type that the owner can choose, and the other tables are access to insert new data, relating to the new user/owner, and selecting the id of the last entry in the table.

Business Layer

The business layer tasks for this component involve mostly checking of the registration inputs of the user/owner. Things like making sure emails are entered correctly and that there are no illegal inputs. The id of the last user to register has to be taken from the database and increased by one to get the new users id. For the owners registration they page where they select their owner type needs the controls that represent owner types created, this is a custom control which is generated dynamically based on the rows in the owner type table. This was done so handle further expansions to this table in the further,

if they wanted to change or add any plans it only has to be done in one location at the database.

User Interface Layer

Figure10. Registration Component User Interface Pages

Figure 10 shows the owner plan select, owner registration form and user registration form user interface pages. The owner plan select page has controls that highlight very visually when clicked on to allow an owner to easily know which plan the have selected. The next and cancel buttons clearly describe what they do and the heading of the pages let the users know exactly where they are and what they're doing. Color is used here to further separate the two different types of accounts.

3.5.4 Login Component

The login component contained the login screen to the system and the business layer code for checking their user inputs. The inputs taken from the user interface were a username and password, and depending on if the inputs matched those stored in the database, the interface was then redirected to the main page of either the owner or user.

Data Access Layer

The database tables accessed by this component were the user, owner, and sub-owner tables. The data access commands were constructed such that the actual password of the user was not sent around the system. The methods took in the username and password, and returned the id if the username and password matched and null if not. A sample of the code for the command to check the password is correct when an input username was their email is below in code sample 1.

```
protected static String checkPassViaEmail(String table, String column, String email,
String pass)
{
        SqlConnection src = new SqlConnection();
        src.ConnectionString = conStr;
        SqlCommand checkPass = new SqlCommand();
        src.Open();
        checkPass.CommandText = "SELECT "+column+" FROM "+table+" WHERE Email=@email AND
                Password=@pass";
        checkPass.Connection = src;
        checkPass.Parameters.Add("@email", System.Data.SqlDbType.VarChar);
        checkPass.Parameters["@email"].Value = value;
        checkPass.Parameters.Add("@pass", System.Data.SqlDbType.VarChar);
        checkPass.Parameters["@pass"].Value = value;
        SqlDataReader r = checkPass.ExecuteReader();
        if (r.HasRows) {
                r.read();
                return r[column]+"";
        }
        else { return null; }
}
```

Code Sample 1. Check Login inputs

The method takes inputs of table, column, email and pass, the table and column inputs are for the distinguishing between user, owner and sub-owner tables. To access the owner table the table input would be “Owner” the column would be “OwnerID”. The final inputs email and pass are the inputs passed from the interface to the business layer.

Business Layer

The business layer tasks for this component are relatively simple, just handling the inputs from the interface. The inputs needed to be checked and sorted into whether it was a sub owner, owner or user trying to log in. Then depending on the results from the data access layer, the business layer will create an encrypted cookie containing the id and then redirect the response to the corresponding main page, or send back an error message to the login page interface. The error message sent back is “Error in ID/Email/Password” which is general enough so that anyone trying to illegally access the site isn’t given more information on their inputs.

User Interface Layer

Figure11. Login Screen User Interface

The user interface for the login screen uses color liberally to provide contrast between the two sections of the site, feedback is given to the user by highlights and tooltips, as you can see when the mouse moves over the owner registration button, the button highlights and a tooltip appears. Each function is clearly labeled making it easy for the user to know what each button will do and what each control is for.

3.5.5 Authentication Component

Data Access Layer

The only table accessed by the data access layer for authentication is sub-owner and this table is only accessed when the cookie is found to contain a sub-owner ID. The command executed on the sub-owner table selects the permission fields that describe what actions a user is able to perform and returns them as a SubOwner object.

Business Layer

The business layer in this component checks the cookie associated with the request, decrypts it and sorts the id into a user, owner or sub-owner. If the requested page is in the users section and the id doesn't belong to a user then the response redirects to the login page of the system and vise versa for the owner section. If the id belongs to a sub-owner it is a little more complicated, the sub-owners permissions need to be retrieved from the database and depending on what they are allowed to do, disable some of the functions of

the page. If the sub-owner somehow requests a page they don't have authorization to use, then the response redirects them to the main page of the owners section.

User Interface Layer

There is no interface layer for this component.

3.5.6 Main Menu – Owner Component

Data Access Layer

There is no data access layer for this component.

Business Layer

The business layer part of this site uses the authentication component to disable sections of the menu if there is a sub-owner logged on who doesn't have full permissions.

User Interface Layer

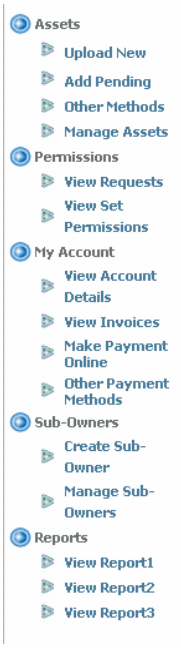

Figure12. Owner Menu

Figure 12 depicts the menu component of the owner section of the site, which gives the owner the opportunity to navigate the different pages of the site. The menu is broken up into sections to allow the owner to easily find the part of the site they are looking for. The blue color scheme of the owners section is again in force here, with most of the test in blue.

3.5.7 Top Header – Owner Component

Data Access Layer

There is no data access layer for this component.

Business Layer

There is no business layer for this component.

User Interface Layer

Figure13. Owner Top Header

The top header is very simple. It just contains the system functions of the site, returning to home, viewing help, contacting Digital Stream, and logging out. The buttons all have a shiny rounded look which makes users think that they would do something, and when the mouse moves over the buttons, they highlight and the mouse pointer changes to the commonly recognized hand pointer, which most users are familiar with as representing a thing you can click on.

3.5.8 Asset Upload Component

Data Access Layer

The data access layer for this component needs to access quite a few of the database tables. The asset table needs to have an asset inserted into it and the owner table needs to be updated for number of assets uploaded and space used. Depending on how the owner has defined their asset, the category and keyword tables may need to have new records inserted. The owner and owner type tables have to be accessed and the limits of the owner's type selected as well as their current usage. All of the current categories an owner has also need to be retrieved and returned to the business layer.

Business Layer

The business layer part of initially begins with retrieving the owners limits and usage and making sure they have room to insert a new asset, and then if so redirecting to the upload page. The upload page contains the current categories of the owner in a drop down list, so these need to be retrieved and put into a list object that the drop down list can handle as a data source. Once the owner has filled out and submitted the upload form, the inputs are

checked for completeness and correctness. Any error found here is returned to the form and it displays the error message to the owner. If the input passes the tests an asset object is constructed. This involves the creation of an image thumbnail, which is then stored in the database. The code for this is shown in code sample 2.

```
Bitmap original = new Bitmap(a.AssetFileStream);
int maxSize = 140;
int thumbWidth = maxSize, thumbHeight = maxSize;
if (original.Width < maxSize && original.Height < maxSize) {
        thumbHeight=original.Height;
        thumbWidth = original.Width; }
else if (original.Width < original.Height) {
        thumbWidth = original.Width * maxSize / original.Height -5; }
else if (original.Height < original.Width) {
        thumbHeight = original.Height * maxSize / original.Width -5; }
Image tempThumb = original.GetThumbnailImage(thumbWidth,thumbHeight, null, new
        System.IntPtr());
Bitmap thumbnail = new Bitmap(maxSize, maxSize);
Graphics g = Graphics.FromImage(thumbnail);
Point p = new Point((maxSize - thumbWidth) / 2, (maxSize - thumbHeight) / 2);
g.Clear(Color.White);
g.DrawImageUnscaledAndClipped(tempThumb, new Rectangle(p,new
Size(thumbWidth,thumbHeight)));
MemoryStream s = new MemoryStream();
thumbnail.Save(s, ImageFormat.Jpeg);
DataInteraction.AssetDA.addThumbnail(s.GetBuffer(), a.AssetID);
a.AssetFileStream.Close();
s.Close();
```

Code Sample 2. Create Thumbnail

This code creates a thumbnail of the image that keeps the same relative dimensions, but with a maximum of 140x140 pixels.

The business layer also identifies any keywords or new categories that have been associated with this asset and calls the insert methods of the data access layer.

The final task is to store the assets data file on the server. The file is renamed to the asset's id and stored in the associated owner's folder.

User Interface Layer

Figure14. Asset Upload

The upload form is grouped into sensible sections, the required definitions – name, description – then the optional keyword, the assets permissions, and finally the brand guidelines to the asset being uploaded. Once the form is submitted a progress bar pops up giving the owner feedback on how long the upload will take, and a progress complete page is shown once the upload has finished, giving the owner the option to add more, view an upload summary or return to the home page.

3.5.9 Asset Management Component

Data Access Layer

The data access layer for this component deals with the same database tables as the upload component. The asset table needs to be accessed as an asset could be updated or deleted. It also needs to be accessed to retrieve all assets belonging to an owner. If an asset is deleted the owner needs to be updated for number of assets uploaded and space used. If the deleted asset contains keywords, then the keywords need to be deleted from the keyword table, however if the asset is edited the keyword and category tables may need to be accessed for editing or inserting. Again all of the current categories an owner has also need to be retrieved and returned to the business layer.

Business Layer

The business layer part of initially begins with retrieving the owners assets and putting them into a table object that can be used by a grid view as a data source. The second task is to handle any changes to assets, this can be deleting, or editing. Deleting is relatively simple, just check that the owner request the delete is the owner of the asset, and then call the delete method of the data access layer. The update method gets a little complicated, firstly the update method takes in the edit form inputs from the user, it then creates an asset object from the asset data which is currently in the database, then depending on if there is a difference between the input data and the stored data, makes an addition to the update command, an example is shown in code sample 3.

```
if (a.Description != Description)
{
        hasChanges = true;
        AssetCommandText.Append("Description = @desc, ");
        AssetUpdate.Parameters.Add("@desc", System.Data.SqlDbType.VarChar);
        AssetUpdate.Parameters["@desc"].Value = Description;
}
```

Code Sample 3. Creating Asset Update Command

Here if the asset stored in the database has a different description to the one from the edit asset form then the command text is added to, and a new parameter is added to the equation.
Another of the business layers tasks is to send the asset's data file to the owner, which needs to be retrieved from the servers file system, and sent in the response.

User Interface Layer

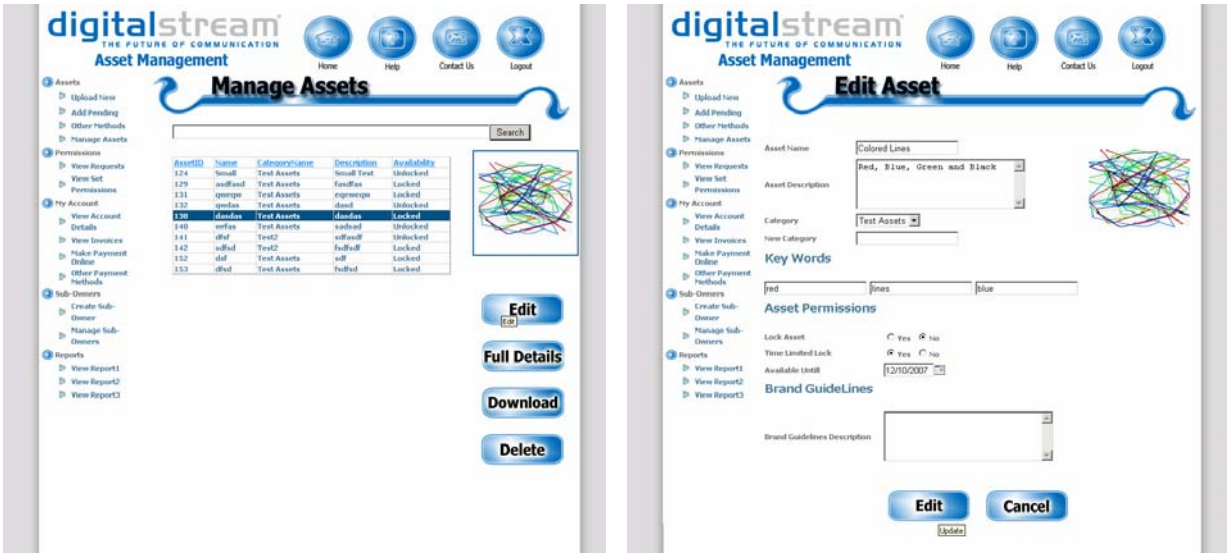

Figure15. Pages of the Manage Asset Component

The manage asset interface page contains a grid view of all the owners assets which they can filter by entering a search term in the textbox above it. The column orders can also be reorder by clicking on the column headers. This may not be immediately apparent to inexperience users but is a feature well known to experienced users. Each row of the grid view can be selected, and on select will highlight and a thumbnail preview will display, to help the user use recognition to know what asset they have selected if the descriptions are not enough. Once an asset has been selected the owner can choose to edit, delete, view or download. If the owner selects delete a confirmation box appear, to make sure they did not click delete by error.
The other page shown in figure 15 is the edit asset page which is a reused version of the asset upload page, any owner that has uploaded using the upload page will be instantly familiar with the design and layout.

3.5.10 Main Menu & Top Header – User Components

Data Access Layer

There is no data access layer for this component.

Business Layer

There is no business layer for this component.

User Interface Layer

Figure16. Users Top Header

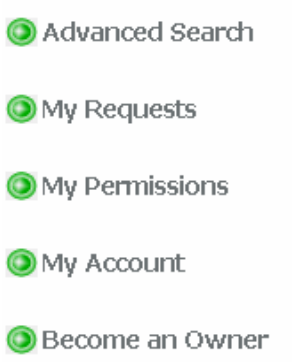

Figure17. Users Main Menu

The designs for the menu and top header for the users section of the site closely follow the same components of the owners section. However the main difference is in the colors use to depict the controls. Instead of the blues use in the owners section, greens were used for users.

3.5.11 Asset Search Component

Data Access Layer

The data access layer for the search dealt with the asset table and the other tables that described the asset, namely owner, keyword, and category. The table access commands joined all these tables together and searched their fields for once that were like an input term. The resulting asset ids were returned.

Business Layer

The biggest task for the business layer at this stage was performing the advanced search. There were so many different ways that the user could want their search performed and the code had to account for all of these.
The other main task was the construction of the custom controls that represented the search results. The user could choose between viewing their search results in a list form or a grid form, and when the user made their selection, the results had to be change

dynamically, this meant taking the assets information and creating the custom control based on this data.

User Interface Layer

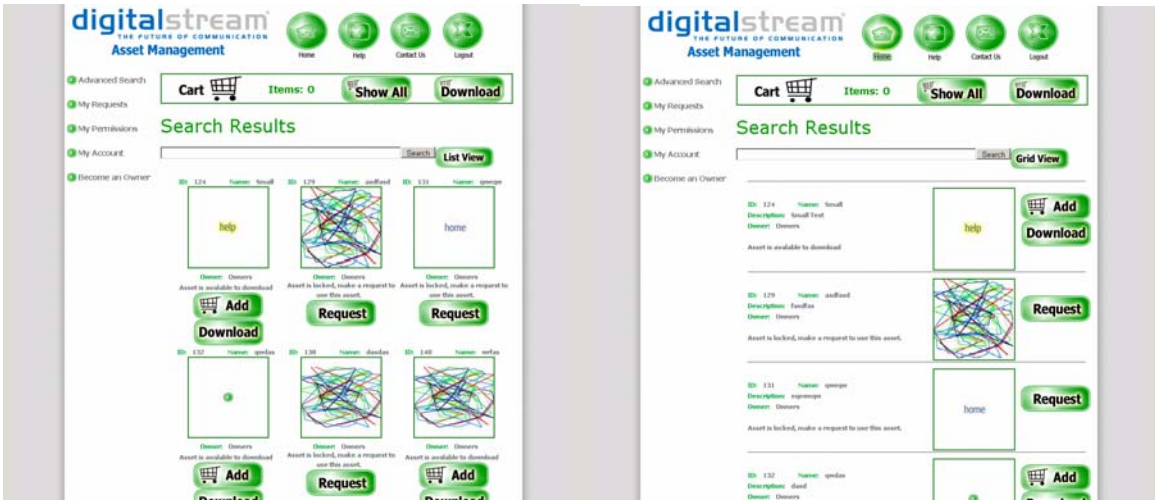

Figure18. Search Results as a Grid and a List

Figure19. Advanced Search Function

Both sets of results come with a thumbnail preview allowing the user to instantly know what there search has resulted in. The List shows more descriptive data but the grid allows a more visual and quicker identification of the asset they want to download.

The advanced search form is broken up into groups, clearly separated by horizontal rule lines, and color. The labels in the drop down list are clear and a user can easily tell what changing the search option will do to the search. The date input for the search assets after or before a certain date option can be input using a calendar that appears when the owner click the calendar icon. The calendar icon is easily recognized for what it is and using a calendar metaphor for choosing a date immediately makes a user comfortable in their task. Having the date chosen from a calendar also reduces input errors from the user where they might type the date in the incorrect form.

3.5.12 Asset Download Component

Data Access Layer

When a user downloads an asset there need to be an entry stored into the download history table, the asset table also needs to be updated with the number of downloads field increased by one. Apart from those two tables however no other data access takes place.

Business Layer

The main function of the download component is obviously sending the file to the user. When the manage assets component was developed, a download function was created to allow the owner to download their own assets. In this case the function can be reused, just changing the check to make sure that the current user has permission to download the asset.

User Interface Layer

There is no interface layer associated with this component.

3.5.13 Cart Component

Data Access Layer

When a user downloads the cart the data access that was carried out when a single asset was downloaded needs to be carried out for each asset in the cart.

Business Layer

Business layer functions for the cart involve updating the cart's cookies – adding and removing assets as the user makes changes – and compressing the files in the cart and sending them to the user. To do the compression, open-source software from Interop, a blog based online developer for .Net [7] was used.

User Interface Layer

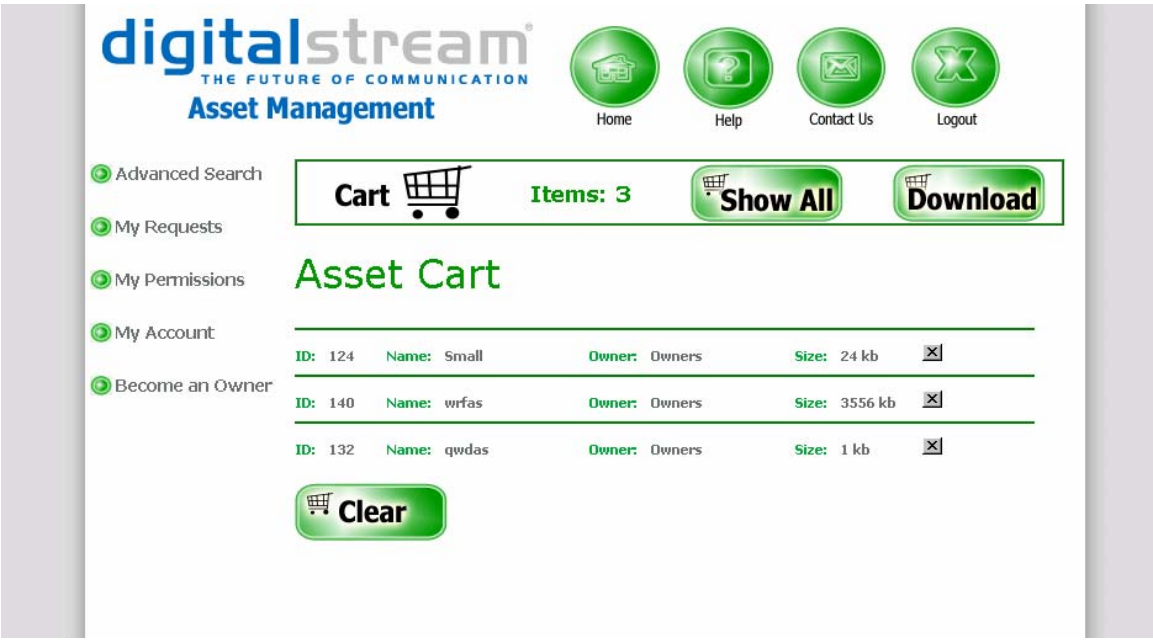

Figure20. Viewing Items in the Asset Cart

Below the top header is the cart summary which is displayed on all of the pages in the user section of the site. It contains an icon symbol of a shopping cart as well as the label "Cart" to make sure users identify that as the cart, and associate the shopping cart symbol with the asset cart. The symbol appears on all buttons that modify the cart, to help the user understand the functions of the buttons.
The view cart page show a list of each of the items in the cart, each of which an be removed by clicked on the "x" on the right of the description. Most users will associate the "x" with deletion, and for those that don't a tool tip appears showing "Remove" to the user. The user also has the option to clear the whole cart at once.

3.5.14 Asset Permission Request Component

Data Access Layer

The only database table accessed by the data access layer for permission requests is the requests table. The command used for this case is the insert command, to add a new row to the table.

Business Layer

The business layer for this component deals with the input checking of the request message and then calling the insert request method from the data access layer.

User Interface Layer

Figure21. Request Permission

The interface for this component only consists of a small popup window that contains the controls that allow the user to create a request message to send to the owner of the locked asset. Once the request is made the asset's status changes to request pending.

3.5.15 Request Management Component

Data Access Layer

The data access layer for request management provides methods for retrieving and removing a user requests. This means it only requires the use of the request table in the database.

Business Layer

The main function of the request management component is to use the data access methods to retrieve all of the current user's requests and put them into a data table that can be used in a grid view. The other function is to call the remove request method, after making sure the current user is the one making the request.

User Interface Layer

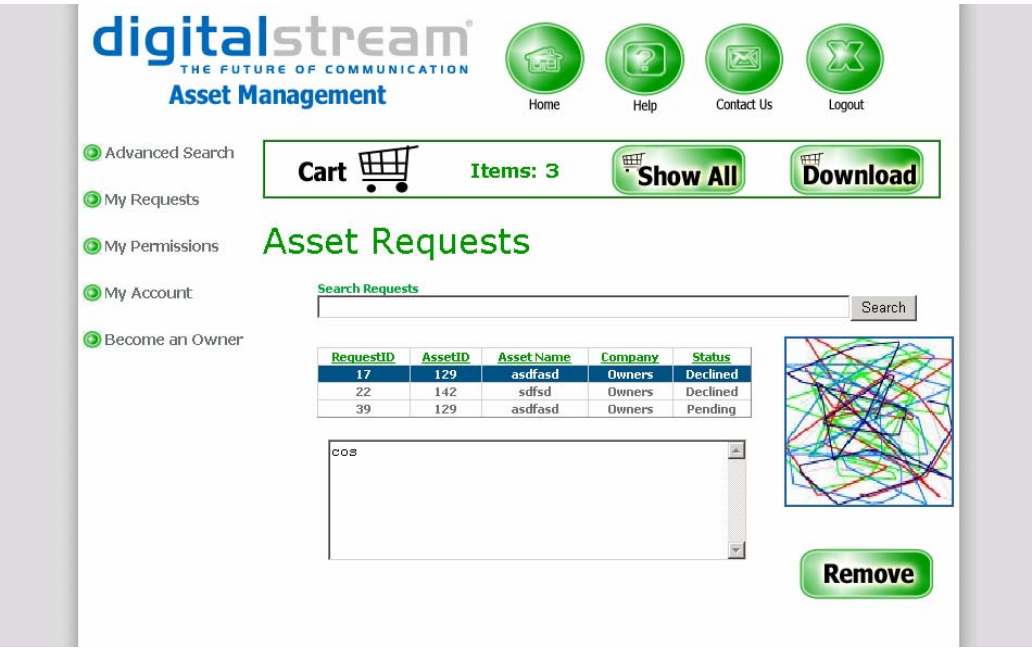

Figure22. Manage Requests

The user interface for the user's manage request page is heavily based on the owner's manage assets page. The request is selected from a grid view that highlights the selected request. If the selected request has been declined, a textbox showing the decline message is shown. The user also has the option to filter the requests in the grid view with a search term and remove any request from the system.

3.5.16 Handling Requests Component

Data Access Layer

The owner uses this component to deal with any requests that have come in from users, either declining or approving them. Firstly the request table needs to have all of the requests relating to the current owner retrieved. Then if the request is declined, the request row in the request table needs to be updated to set it to not live, and set the decline message to the users input. If the request is approved then the request needs to be removed, and a new entry inserted into the permission table.

Business Layer

The business layer tasks for handling requests firstly deal with the retrieval of the owners requests, and then with the handling of the requests. Most of the functions are just

passing information through to the data access layer but, if the owner sets a limitation date for the permission, then this need to be check to see if it is a valid date. If the owner declines the request then their request message is also checked for validity before forwarding to the date access layer.

User Interface Layer

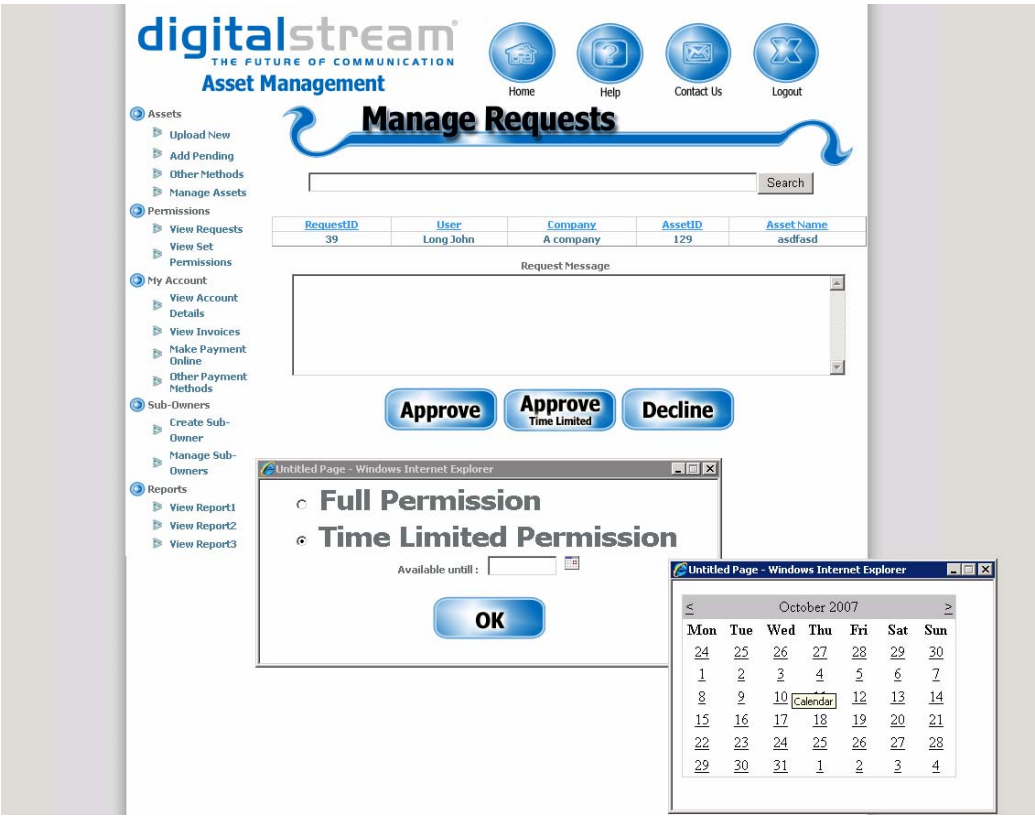

Figure23. Handling Requests

The user interface is again based on the manage assets interface, the owner must first select a request from the grid view then they can perform actions upon it. This consistency helps keep user comfortable in the system. If the owner chooses time limited approval, the options are consistent with options they have when adding or editing an asset.

3.5.17 Permission Management Component

Data Access Layer

Data access for the permission management component only requires the use of the permission table. For retrieving all the owner's permissions, removing any of their permissions, and finally for updating their permissions.

Business Layer

The business layer doesn't have much to do in this component, mainly just forwarding data between the data access and the user interface. One thing that is required here is validity check of any dates the owner sets the time limited permissions to.

User Interface Layer

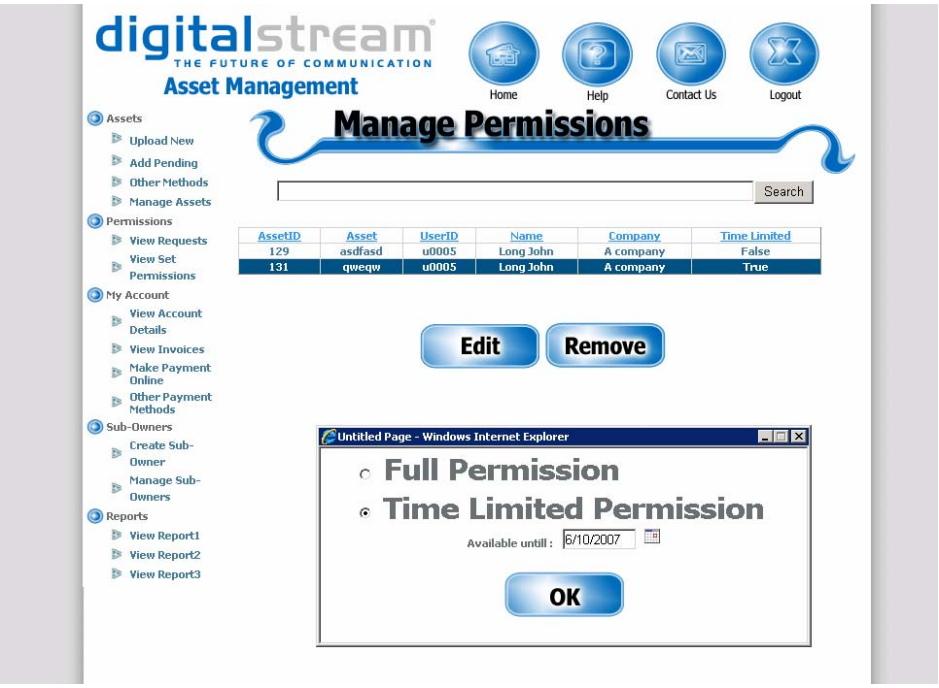

Figure24. Permission Management

The permission management interface is completely based on the handle requests and manage permission interfaces, only differing in what options the owner has when they select a permission. If they click remove then a confirmation box will come up to make sure they didn't click remove by accident.

3.5.18 Messaging Components

The messaging components are for creating the information that is displayed to the owners and users when the first log into the system, alerting them of any actions that have happened since their last log in that require their attention. This can be new request for owners, requests responses for users, or a system message from administration.

Data Access Layer

Messaging components involve almost all tables of the system. To get the users messages, the user table needs to be accessed to get their last login date. The request table is used to retrieve any new decline responses, the permission table to retrieve new permission, and the system messages table for new system messages.
To get owner messages, the last login date needs to be retrieved from the owner table. Any new requests need to be gotten from the request table. Owner type limits need to be gotten from the owner type table and usage needs to be gotten from the owner table. System message

Business Layer

The tasks associated with the business layer for this component are to retrieve the data required for the messages and put in into a form that can be displayed to the user or owner. The involves some calculation for the limits and usage of the owner, and the construction and display of a pie and bar graph to make it easy to visual how much of the space limit the owner has used up.
The rest of the task is turning the new requests, responses, and system messages into simple string messages. An example of this is shown in code sample 4 below.

```
public static String[] getNewMessages(String ownerID)
{
	ArrayList a = new ArrayList();
	//Approaching UsageLimit
	bool limit = false;
	double[] spaceUsage = DataInteraction.OwnerDA.getSpace(ownerID); //[used][limit]
	if ((spaceUsage[0]/spaceUsage[1])>.9) { limit = true; }
	int[] numberUsage = DataInteraction.OwnerDA.getNumber(ownerID); //[used][limit]
	if ((numberUsage[1]-numberUsage[0])<2) { limit = true; }
	if (limit) { a.Add("Warning you are approaching your upload limit, consider
		deleting some old assets or changing you account to a higher limit."); }
	//OverDue Payment
	if (DataInteraction.OwnerDA.paymentOverdue(ownerID)) { a.Add("Warning: You have a
		account payment overdue please pay as soon as possible."); }
	else
	{
	//Payment Due in 7 days
	if (DataInteraction.OwnerDA.paymentDue(ownerID)) { a.Add("You have a account
		payment due within the next 7 days."); }
	}
	//General Messages
	String[] sysMsg = DataInteraction.OwnerDA.getSystemMessages(ownerID);
	for (int i = 0; i < sysMsg.Length; i++)
	{
		a.Add(sysMsg[i]);
	}
	return (String[])(a.ToArray(typeof(String)));
}
```

Code Sample 4. Get New Owner Messages

User Interface Layer

Figure24. User Messages

The user interface shown in figure 24 is the main page that a user sees when they first log into the system, the new responses and messages clearly identified to them.

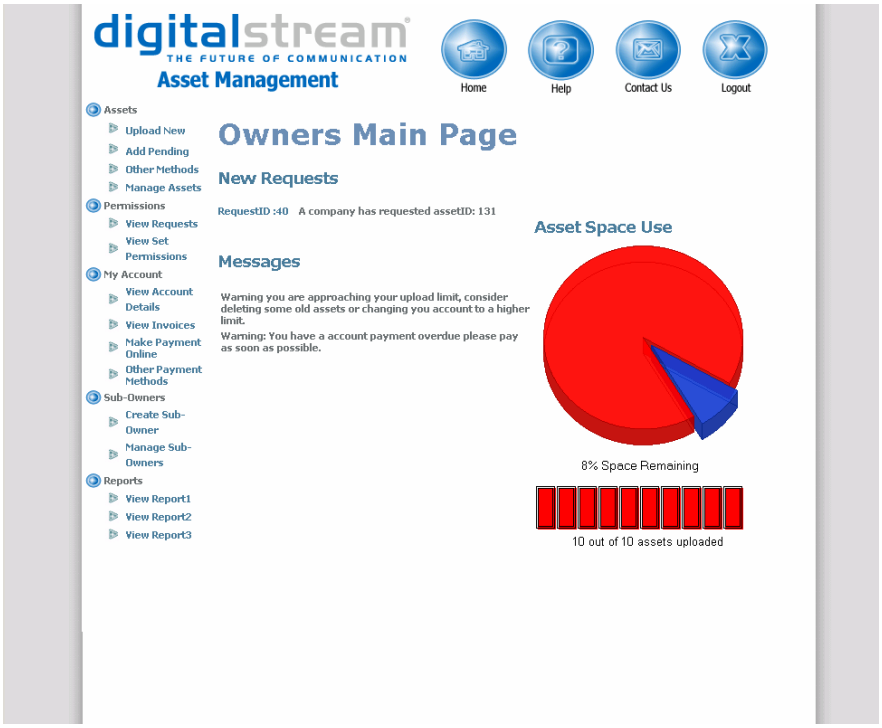

Figure25. Owner Messages

The interface in figure 25 is the main page for owners when they first login to the system. New requests and Messages are shown clearly on the left, and on the right show a graphical representation of the owner's current usage in relation to their limits, with red representing used space and blue free space.

3.6 Testing Phase

Testing during the construction of the components was done only based on the input and output of the components, if it took in correct input and gave correct outputs, and then it worked. If it didn't accept incorrect input and return an informative error message able to be displayed in the user interface then it also was said to be working. The output from the component could be a string or Boolean value but was most often an addition or edit of a data row of a table in the database. The output then was tested by directly checking the information in the database, making sure it matched what was expected. This type of testing is called black box testing, and is the easiest type of testing to perform on a system.

3.6.1 Usability Testing

Usability testing evaluates product designs and implementations to provide feedback to that leads to improvements, bug reduction and removal, and validate that the system is and will conform to fulfilling end users goals. [5]
There are four usability objectives for the usability of a system.

- Usefulness – where users can or cannot successfully complete a task.
- Effectiveness – where users are tested if they can complete tasks within a set period of time
- Learnability - where are a certain amount of training users obtain a predefined level of product knowledge.
- Attitude – where the satisfaction of the users after using the system is measured.

The main problems with usability are that it requires larges amounts of time and cost it set up. You need to find a set amount of tests users that fit the demographic of the systems end users, this amount is normally around 5-10 users. These users all have to be paid and set up in environments that the system will be used it.
However, even when you take the problems into consideration, the overall results from usability testing cannot be ignored. If the money spent on usability testing reduces the future downtime and required upgrades to the system, then overall the company will be saving money.

Once this project is fully complete, usability testing will need to be done before it is released into the market. As Digital Stream has many customers wanting to use this sort of system already, finding test users will not be a problem.

4 Conclusion

This project involved the development of quite a large web based system, with a lot of user interface that was required to be developed. The project began with extensive research into digital asset management systems, which gave a good base to understanding what was required and how to develop the system. Initially more time was spent on analysis and design of the system as a whole, as the interface design were expected to be done by other workers in Digital Stream, however this did not end up happening and having to research interface design and design the pages myself created set backs that didn't allow the project to progress at the expected rate. However learning and experimenting with interface design gave good experience, especially with being able to implement my own designs. The system still needs to have finishing touches applied and extensive testing done before it can be marketed and sold as a commercial product.

5 References

[1] J A Hoffer, J F George, J S Valacich. *Modern Systems analysis and design 3rd Edition*. Prentice Hall International

[2] E-See. *Brand Asset Management.* http://www.e-see.com

[3] Original Image – *Digital Asset Management.* http://www.originalimage.com/

[4] J L Whittey, L D Bentley, K C Bittman. *System Analysis Design Methods 6th Edition.* McGrawHill Irwin

[5] T Mandel. *The Elements of User Interface Design.* Wiley Computer Publishing

[6] J Johnson. *GUI Bloopers.* Morgan Kaufmann Publishers

[7] D Chiesa. *All About Interop.* http://blogs.msdn.com/dotnetinterop/default.aspx

[8] J J Odell. *Advanced Object-Oriented Analysis and Design Using UML*. SIGS Books