CS 5551 Project Plan

Adam Carter
Class ID: 9, Project Group ID: 1
CS 5591 – Spring 2015

I. Introduction

Board Gaming, a hobby that consists of the collection and playing of Board and Card games, is a hobby community that has been growing steadily in recent years. As hobby communities grow, technology can be a valuable asset to help individuals and communities organize and manage collections. In the Board Gaming domain, because of the difficultly of compiling and updating a single source of content, only one website, BoardGameGeek[1], has been able to compile anything approaching a complete list of games belonging to this hobby. Their site provides a number of resources that provide information about board and card games, but the site suffers from a number of usability issues and functional limitations. The aim of this project is to enhance the collection management experience by leveraging this index of information, connecting that data with marketplace retailers, and giving users tools to improve their collection management experience.

II. Project Goal and Objectives

Overall Goal

My objective for this project is to create a set of web services capable of managing a user's collection and providing a single point of reference for personally relevant board gaming-related information. I plan to create a single tool that will provided services that should join together all of the currently separate aspects of collection management to improve the hobby experience. I additionally plan to provide tools to help customize game meta-data to user's preference, instead of relying on existing static meta-data. For this project to achieve its stated goals, many of these tasks must be tailored towards the mobile user.

Specific Objectives

For anyone involved in this hobby, there are several tasks and questions that come up repeatedly. How many games do I actually own? How many games do I have that can be played with six players? Where can I buy this game? When did I play this game last? These are examples of questions this tool should be able to answer that are difficult to answer currently for most hobbyists, especially those with large collections. In order to be able to serve these needs, the system needs to possess the following capabilities:

- Stay up to date with the newest game information from BoardGameGeek.
- Distinguish between Base games and Expansion titles.
- Allow users to add to, remove from, and search through their personal game collection.
- Generate play recommendations based on user-defined criteria (i.e. 'Games that can be played with six players' or 'Games that play in less than 30 minutes')
- Access common retailer information and compare prices to identify where a game can be purchased.
- Allow users to build a 'Wish List' of future purchases, and notify them when titles become available.
- Record information about when a game in the user's collection was played.
- Present users with quick access to game related material, such as Game Manuals, 'How to Play' videos, or game reviews.

Some of these tasks exist in some form through current services, but no single tool exists that is capable of providing all the services indicated above. The goal of this project is to develop a set of web services and user tools to facilitate all of the proposed criteria.

Significance

While comparable hobby communities such as baseball card or sports memorabilia collectors have a number of resources available to them that fill this role of collection management, the board game community is severely lacking in such tools. Only one website exists – BoardGameGeek[1]- that even attempts to catalog all board games as they are released and provide users tools for collection management. According to 2013 statistics, this site has over 780,000 users and received 570 million page views during 2013 [1], which demonstrates the size of a potential user base. Despite the heavy traffic, this site has become large, unwieldy, and has a notoriously poor search engine. No other gaming website generates even a fraction of the web traffic or approaches the scope of content available. Numerous small niche projects have been started to tackle single aspects of this problem, none of which have gained any wide traction in the community. This means there is room for a more comprehensive tool that can bring together many of the strengths smaller projects have attempted to provide. Additionally, BoardGameGeek hosts a vibrant trading community, but many hobby retailers do not share information with sites like BoardGameGeek. Any tool that can successfully link game entries with retail providers will significantly improve the purchasing experience.

III. Project Background and Related Work

As mentioned above, there are very few projects that have explored the collection management space for Board Game hobbyists. Boardgamegeek.com released iOS [3] and Android [4] apps that aim to provide limited or incomplete access to all the features of the website, but they both suffer from poor implementation, incomplete functionality, and low adoption rates. The iOS version currently holds a rating of only 2 out of 5 stars, and neither version reaches anywhere close to the same audience of the website.

BoardGameGeek also released an XML-based REST API [5] for accessing read-only content found from their site with the hope of inspiring hobbyists to develop tools on their own, but very few applications exist that attempt to do more than simple tasks with that data. There were two interesting examples I found. The first is a relatively new website called Board Game Menu [6]. With this site, you provide your BoardGameGeek user name, answer a few questions about what type of game you want to play, and it generates a random 'menu' of games that fit the criteria as recommended games. This is a feature I consider to be important for any collection management tool to provide. Another standalone application called WhatToPlay Deluxe [7] provides a much more comprehensive set of interactive questions designed to choose a game from your BoardGameGeek collection to play. This application has not been worked on in almost three years, and the lack of support and compatibility issues have prevented it from expanding further.

A sample of tools found for collection management in other hobbies gives an idea to the variety of tools that exist in this space already. From open-source tools for music collectors [8], movie enthusiasts [9], recipe collectors [10], and LEGO enthusiasts [11] to free apps for book collectors [12], coin collectors [13], or baseball card collectors [14], these tools share several common features. First, they rely on an external source of truth, such as IMDB for movies or US Mint reports for Coin Collectors, to help define the collection. Second, they provide the user with means to organize and sort their collections. Third, they access external price sources or other information to help determine the value of one's collection. Fourth, many of these tools provide ways to share the collection with others or make recommendations

based on their current collection. These are all elements I believe are necessary to make a complete hobby collection management system viable.

IV. Proposed System

1. Requirement Specification

Requirements

i. The system shall allow users to create personal accounts to access the system.
ii. The system shall allow users to build a personal game collection.
iii. The system shall allow users to select any game from the BoardGameGeek index to add to their collection.
iv. The system shall allow users to search their collection for games based on common criteria.
v. The system shall allow users to access retailer information for games available for purchase at their respective sites.
vi. The system shall allow superusers to map board game entries to retail offerings when unable to be mapped automatically.
vii. The system shall allow users to build a list of games they desire to purchase soon.
viii. The system shall allow users to add or view supplemental game content such as game manuals and related video content.
ix. The system shall allow users to edit subjective game definitions like game weight.
x. The system shall be able to distinguish between Base, Expansion, and Collectible games.
xi. The system shall be able to detect new offerings from potential retail providers.
xii. The system shall allow users to be notified of selected system events.

Workflow Analysis

There are three key workflows categories involved in this project.
First, the automated scanning requirements of external APIs. These tasks will consist largely of writing HTML or web service requests to outside data sources, transforming that data, then writing the necessary output to our database. Because there are few decisions to be made here, Activity Diagrams are not of any value to these workflows.

Second, the superuser requirements for helping ensure collection database integrity. This task should involve inspecting and confirming that BoardGameGeek data is being interpreted correctly, and that retail entries are being correctly mapped.

Third, the common workflows performed by users. Each of these tasks will correspond to simple tasks the user will be required to perform. These tasks will be outlined by user stories as the project proceeds, and the corresponding Activity Diagrams are so simple as to not have any value this early in the development process.

Technological and Architectural Requirements

First, I will discuss the technological requirements. This service will need to access multiple outside sources of information to perform the required tasks. Some services (YouTube links, Amazon requests) already provide APIs to facilitate access. Others do not.

Our primary source of data, BoardGameGeek, has a largely read-only XML interface, meaning our web service will require the correct XML schema and XML parsing capability to interpret that information.

Some vital data, such as game pricing, is only available from some key sites in static HTML pages, meaning to mine that data, HTML parsers will be required to mine the correct content from the requested pages, along with an understanding of the appropriate tags used to hold that information.

As the implementation of this project will mandate the creation of RESTful services, Java will serve as the development language of choice due to the ease of access to REST frameworks like Spring or Jersey.

In addressing Architectural Requirements, we must consider the required system layers first. Because we will be managing our own collection information for users, we require a database for hosting our information. Though any relational or NoSQL database can fill this role, MongoDB will be used for this project.

Second, a web service will be required to manage user access and personalized content access. This system must be a RESTful service, and as such will conform to the REST architectural style.

Third, two different user clients will be required, based on user roles. Administrative users will not possess personal collections, but will manage relationships between outside data sources. This is necessary since many of the retail systems in particular are not capable of pushing out data updates, so this data will require pulls, parsing, and manual conflict resolution or assignment when mapping relationships are not immediately discernable. This load may also be data intensive, especially in early stages of the project, in which case a native client might handle such data loads better.

The user client will manage all the collection aspects of this project. A traditional web client should be created that can facilitate this, with a consideration towards mobile usage.

Additionally, time permitting, a native mobile application can be considered to also satisfy that need, but is considered a low priority at this time.

2. Framework Specifications

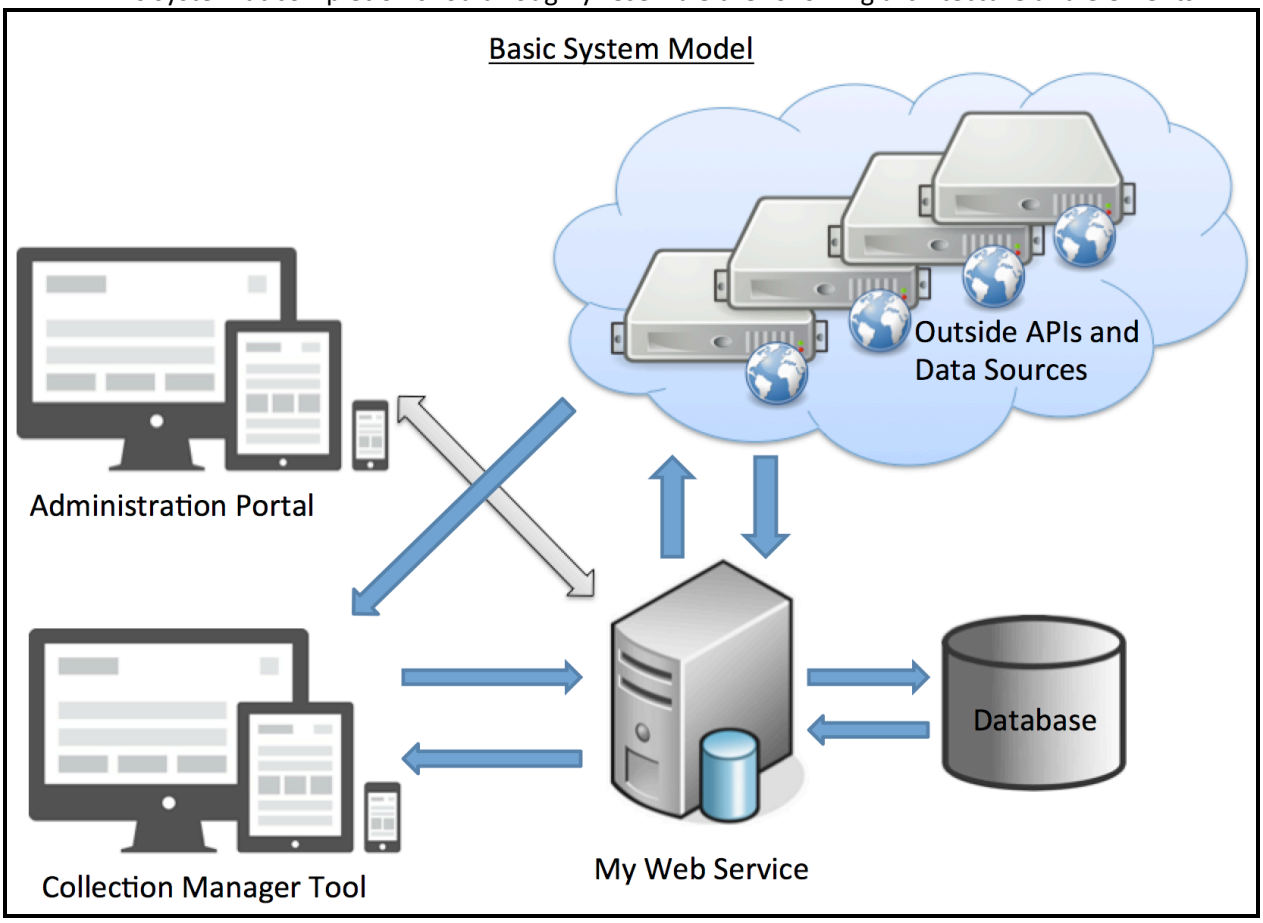
As a system model, my system will strongly resemble a classic three-tier architecture, with the core elements being database, service, and client. The primary deviation from that model will be the reliance at both server and client levels on external APIs to provide information.

Because a majority of our data sources do not ‘push’ data, meaning there is no notification system to notify when items are added or updated, we will need to proactively ‘pull’ desired data and record that data in our system appropriately.

As discussed previously, our system will leverage the REST architecture to design a set of RESTful services capable of supporting our collection users as well as administrative users.

Since users will not generally be logged in to the system often, we cannot rely on a traditional publish-subscribe model, and will instead need to record notifications to present users on system login.

This system at completion should roughly resemble the following architecture and elements:



3. System Specifications

Existing Services

The following represent existing services/data providers that are currently envisioned for this project:

BoardGameGeek XML API: http://www.boardgamegeek.com/xmlapi [5]

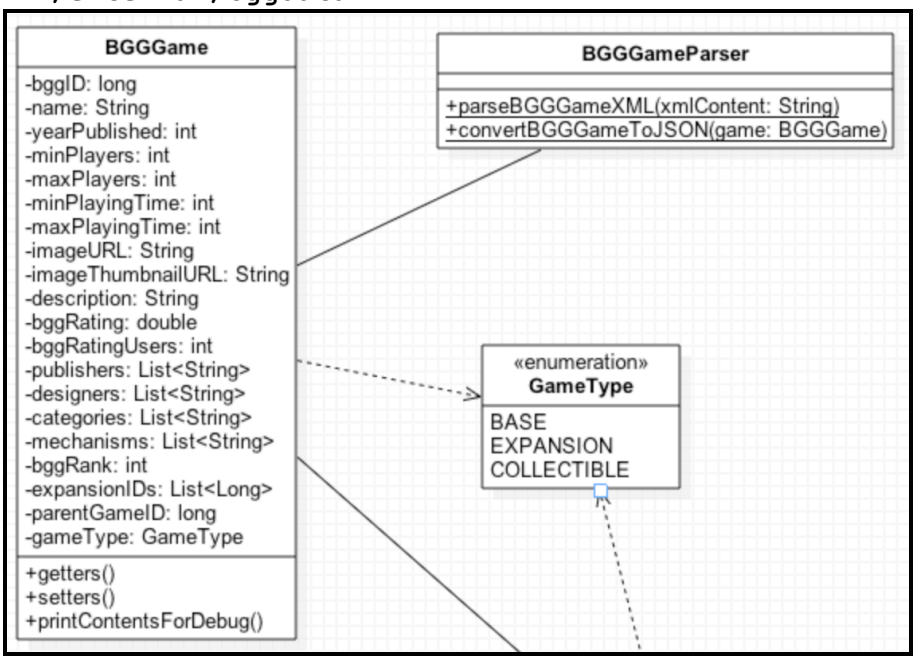
The following represent services that are likely to be consumed by this project

- `/xmlapi/search`
  API to search BGG database by Game Name
- `/xmlapi/boardgame/<gameid>`
  Retrieve detailed information about a particular game. Useful parameter settings include `/<gameid>?stats=1`, which will return statistics about the game, and `/<gameid>?historical=1&from=<startdate>&to=<enddate>`, which will give historical ranking information about the game requested
- `/xmlapi/collection/<username>`
  This will be a useful way of prepopulating a user's collection based on data already provided to BoardGameGeek. There are numerous parameters to narrow the provided list.

Amazon Item Lookup by ASIN/Title: http://webservices.amazon.com/onca/xml?Service=AWSECommerceService [15]
This service will be used to retrieve price information on game titles available from Amazon. This process still needs to be researched more fully, though I have obtained the required Access Key ID and Secret Key data required to interact with Amazon Web Services.

Partially Available Services
These are defined as web data sources that do not provide an API for access, but contain required data. This currently consists of the following:
CoolStuffInc.com: http://www.coolstuffinc.com/p/<gameid>
This is one of the top volume board game retailers. Its site does not expose price data through a consumable API. It does, however, provide a consistent unique URL mapping for a game, and with that data, we will need to write an HTML Parser that can extract the required data elements we require
MiniatureMarket.com: http://www.miniaturemarket.com/catalog/product/viewid/id/<gameid>
This is another of the top volume board game retailers. It also does not provide price data through an open API, but possesses multiple URLs which access the same page data. We will require an HTML parser to extract the required data from this data.

New Services
- **`/external/bggdata`**

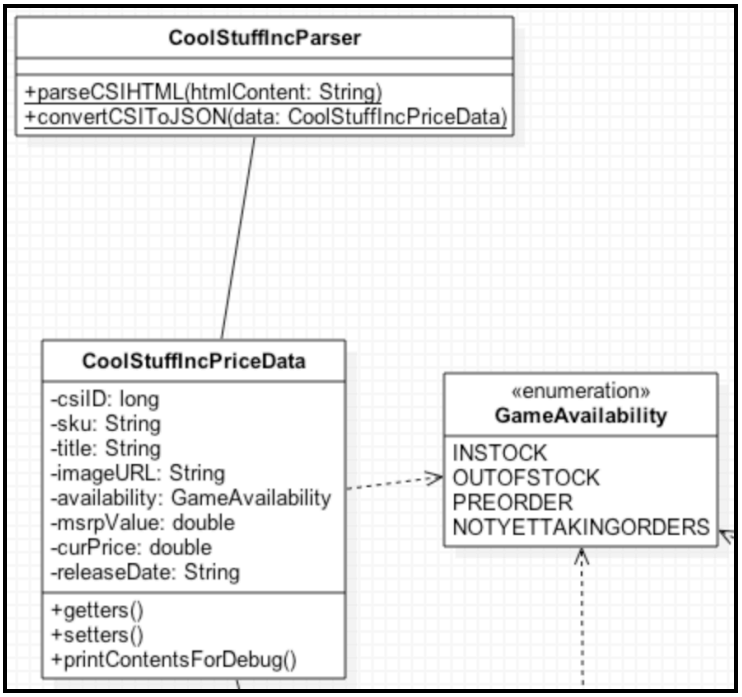


This service needs to be able to handle reads and writes to the database for data coming from the BoardGameGeek XML API. The following operations should be supported:
GET – Parameters should indicate whether to get from BGG or from the database.
PUT – Update database record with updated BGGGame object
POST – Add BGGGame object to database
DELETE – Delete BGGGame object from database by `bggid`. Do not envision this being used.

While XML parsing is possible client-side, this GET request should be the preferred means of requesting this data. Chief parameters should consist of `/bggdata?bggid=<gameid>`. GET operations should additionally support `/bggdata?bggid=<gameid>&source=<bgg|db>`.

This is an essential operation. BoardGameGeek data will form the spine of our data set. This system will also occasionally update the stored content to reflect updated statistics for games. The PUT, POST, and DELETE options should only be available to the Admin users.

- **`/external/csidata`**



This service needs to be able to handle reads and writes to the database for data coming from the CoolStuffInc website. The following operations should be supported:
GET – Parameters should indicate whether to get CoolStuffIncPriceData live from website or from the cached database record.
PUT – Update database record with updated CoolStuffIncPriceData object
POST – Add CoolStuffIncPriceData object to database
DELETE – Delete CoolStuffIncPriceData object from database by `csiid`. Do not envision this being used.

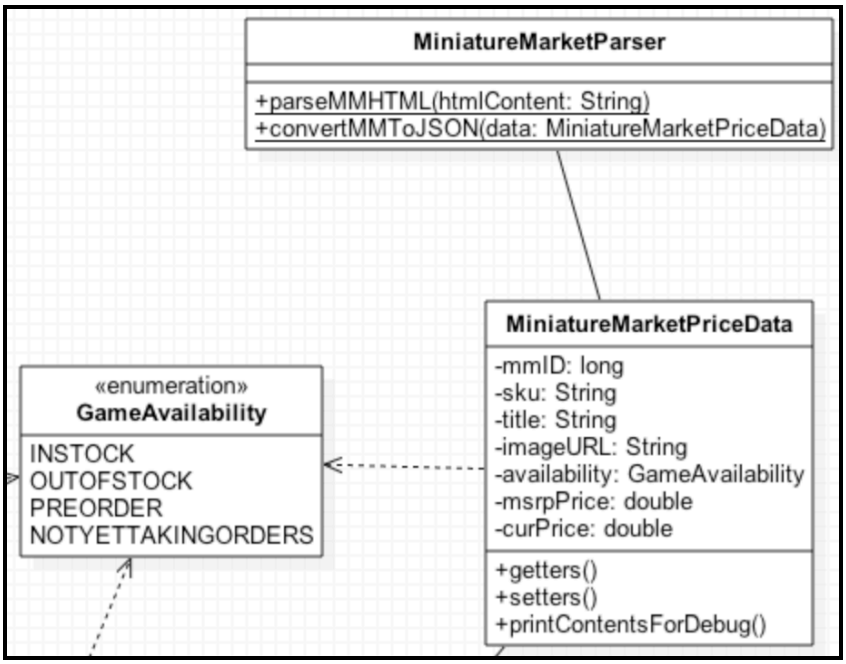
Because of the HTML parsing required, all live requests for this price data must come through this GET request.

Chief parameters should consist of `/csidata?csiid=<gameid>`. GET operations should additionally support `/csidata?csiid=<gameid>&source=<csi|db>`.

This is an essential operation. This is one of the two pivotal price data sets requested by the system. This system will also occasionally update the stored content to reflect updated price and availability information for games. The PUT, POST, and DELETE will most likely be used by

Admin users, but there may be instances when updates to this data are pushed by collection users.

- **`/external/mmdata`**



This service needs to be able to handle reads and writes to the database for data coming from the MinatureMarket website. The following operations should be supported:
GET – Parameters should indicate whether to get MiniatureMarketPriceData live from website or from the cached database record.
PUT – Update database record with updated MiniatureMarketPriceData object
POST – Add MiniatureMarketPriceData object to database
DELETE – Delete MiniatureMarketPriceData object from database by `mmid`. Do not envision this being used.

Because of the HTML parsing required, all live requests for this price data must come through this GET request.

Chief parameters should consist of `/mmdata?mmid=<gameid>`. GET operations should additionally support `/mmdata?mmid=<gameid>&source=<mm|db>`.

This is an essential operation. This is one of the two pivotal price data sets requested by the system. This system will also occasionally update the stored content to reflect updated price and availability information for games. The PUT, POST, and DELETE will most likely be used by Admin users, but there may be instances when updates to this data are pushed by collection users.

While the data object for MiniatureMarketPriceData is similar to CoolStuffIncPriceData, they will not share common data since the underlying format and thus available data elements may change. By leaving them as unrelated classes, we allow for cleaner maintenance going forward.

- **`/external/awsdata`**

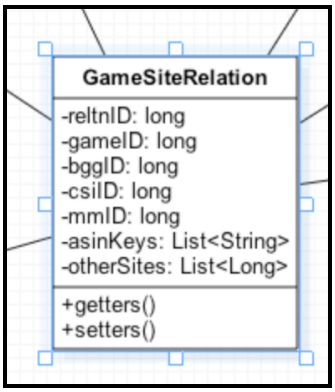
This service needs to be able to generate a properly formatted request to the Amazon Web Service. Details of this process need to be worked out. An index of valid ASIN terms needs to be created so that we don't need to dynamically search for the correct ASIN each time. The following operations should be supported:
GET – Parameters should indicate whether to get AmazonPriceData live from website or from the cached database record.
PUT – Update database record with updated AmazonPriceData object
POST – Add AmazonPriceData object to database
DELETE – Delete AmazonPriceData object from database by `asinKey`. Do not envision this being used.

Chief parameters should consist of `/awsdata?asin=<ASIN Key>`. GET operations should additionally support `/awsdata?asin=<ASIN Key>&source=<aws|db>`.

Unlike some of the other vendors, the possibility exists that a single game title may possess multiple ASIN key values. These would correspond to differing print runs of the same version of a game.

This operation is one of the most complicated, largely due to the security formatting required to successfully interact with Amazon's web service. It is also not as critical to the system, and can be left for later iterations.

- **`/gamereltn`**

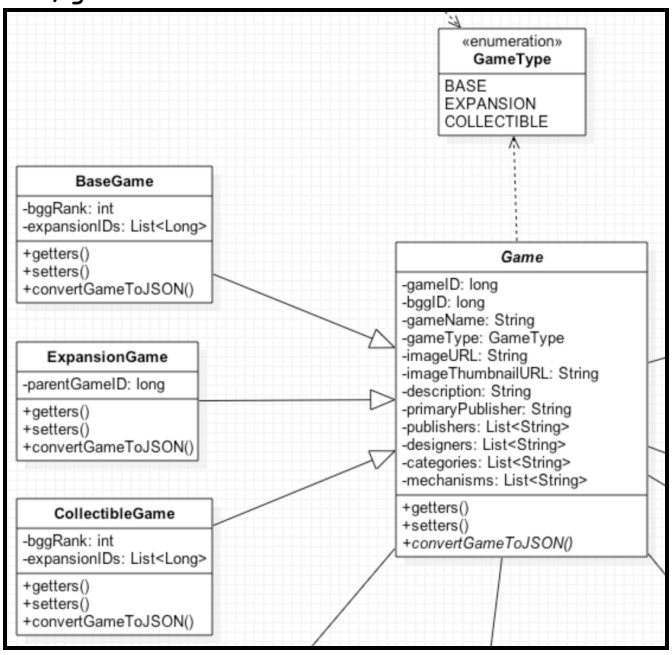


This class is primarily for linking all the various price sources together for reference purposes. This work will be initially built out by Admin users, but can have select additions made, primarily in the otherSites list, by collection users. The following operations should be supported:
GET – We should be able to request GameSiteRelation objects either by reltnID if we know it or by gameID (the most common case)
PUT – Update database record with updated GameSiteRelation object
POST – Add GameSiteRelation object to database
DELETE – Delete GameSiteRelation object from database by `reltnID`. Do not envision this being used.

Chief parameters should consist of `/gamereltn?gameid=<gameid>`. GET operations should additionally support `/gamereltn?reltnid=<reltnID>`.

This operation is critical in terms of being able to help understand how the various sites link together, but not complicated. It should largely be a one-time evaluation effort to build/update the object, but read frequently.

- **`/game`**

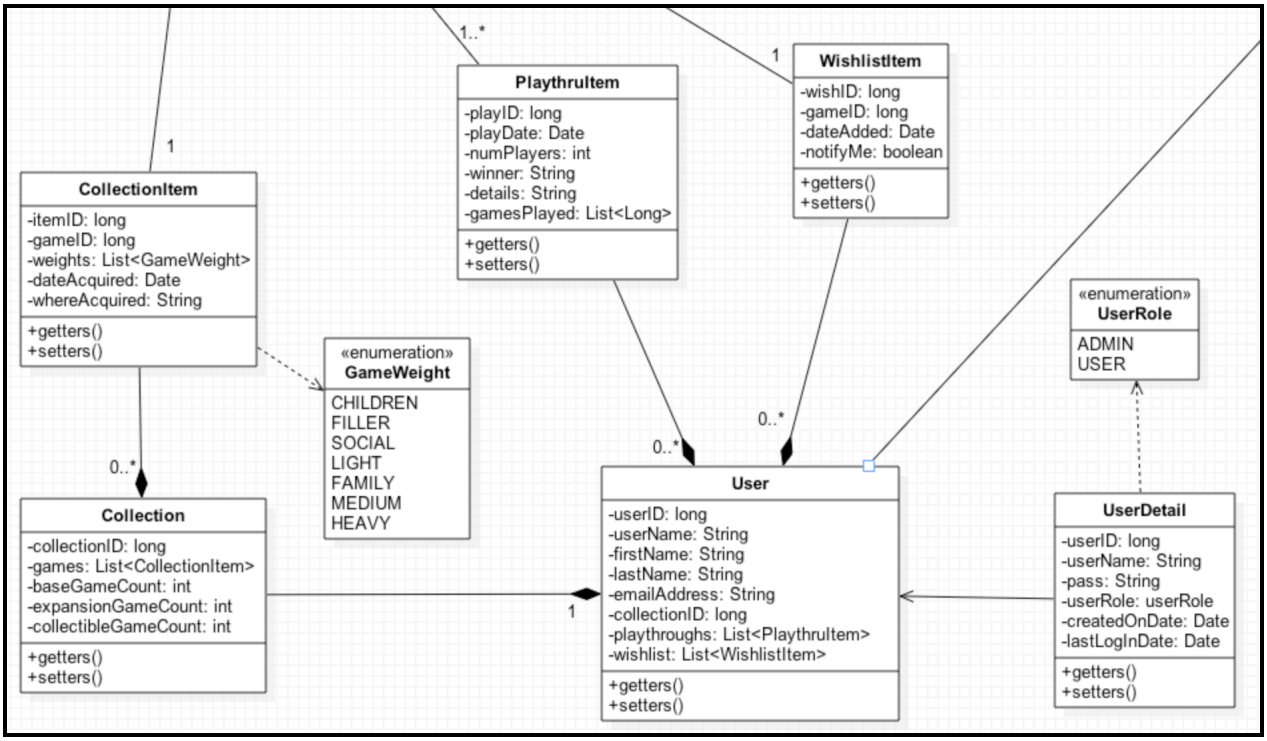


This class is the primary data element for collection users. This is the object that will represent a game to them. We will subclass out games to differentiate between Base, Expansion, and Collectible game types, though each will be stored under the Game element in the database with the differences permissible through JSON composition. The following operations should be supported:
GET – This is a pretty basic read request by the `gameid`.
PUT – Update database record with updated BGGGame object
POST – Add BGGGame object to database
DELETE – Delete BGGGame object from database by `bggid`. Do not envision this being used.

Chief parameters should consist of `/game?gameid=<gameid>`. PUT, POST, and DELETE operations should only be performed by Admin users.

This is an essential operation. By and large this class is a direct mapping of BGGGame fields, but some additional elements have been added, and some, such as primaryPublisher, may require an Admin user to determine which of the provided publishers is the 'true' publisher of the game (many BoardGameGeek entries also credit international distributors in this category). Since this is such a foundational piece of the project, the work to generate these objects should be performed early in the development process.

- **`/user`**



While none most of the other classes listed here aren't directly impacted by this service, this seemed the most logical point to display how the other services relate to the User. Central to the User tasks are `User`, `UserDetail`, and `UserRole`. A user has all the relevant personal information, and contains links to other key items, such as his collection (via collectionID), his recorded playthroughs, and his current wishlist. The following operations should be supported:

GET – This is a pretty basic read request by the `userid`.
PUT – Update user record with provided content updates.
POST – Create a new user.
DELETE – Delete a User object from database by `userid`. Do not envision this being used.

Chief parameters should consist of `/user?userid=<gameid>`

UserDetail information should be kept restricted. There may be some shuffling of fields between UserDetail and User as this workflow becomes more fully realized.

This workflow is obviously essential, but could prove to be rather complicated. Research is still required to determine how to protect user information stored in the UserDetail field that should not be publically visible. Validation work will be required to ensure that all provided fields are correct before registering users. Additional effort may be required to determine how to manage the difference in Admin vs Collector roles.

- **`/search`**

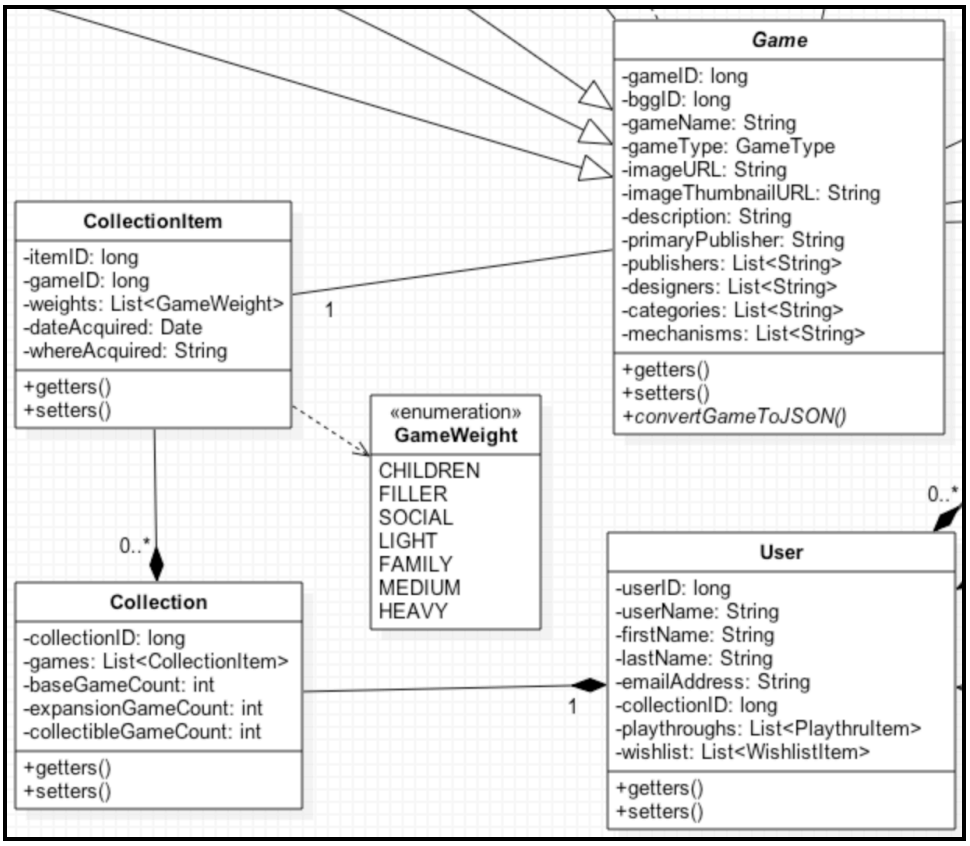
This service does not present a Class Diagram, as it will simply be reusing elements of other services. This service should at a minimum provide search capabilities across game names or other dynamic criteria, independent of a user's collection. This operation should be read-only, meaning PUT and DELETE operations will not be supported. POST operations should not be allowed for a truly RESTful service, but I am leaving this option open to allow for potential RPC consumption, which only uses the POST operation.

The parameter list is still being defined, but may include the following elements:

```
/search?gameid=<gameid>
/search?gamename=<Game Name String>
/search?index=<game|bgg|csi|mm|aws|other>
/search?orderby=<sortable criteria, tbd>
/search?publisher=<publisher string>
/search?designer=<designer String>
/search?category=<category String>
/search?mechanic=<category String>
/search?playerCount=<number of players>
/search?year=<year published value>
/search?limit=<limit results to this many hits>
```

This operation is important, but is definitely second tier in terms of priority. Basically all the multitude of parameters are simply translated into generating a query and returning the appropriate result set. The typical return case will return a List of Game objects, but there might be cases, particularly for the Admin user, where they want to return lists of qualifying entries from the cached results sets. Overall, I don't see this operation posing a great deal of complexity outside initial query construction and execution, though name searches may need to be investigated for improving result accuracy.

- **`/collection` AND `/collection/item`**



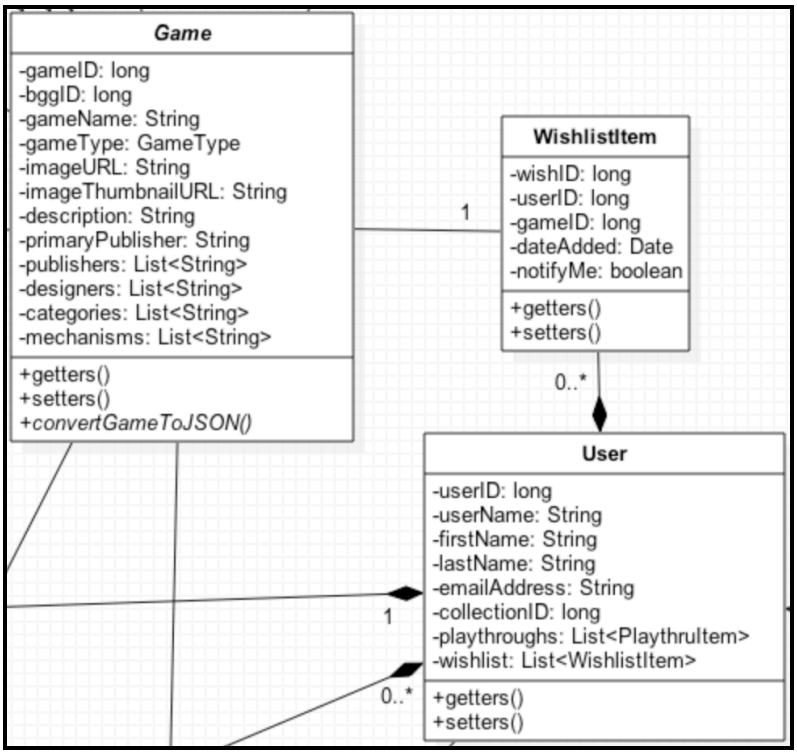
This service should fetch the collection of a given user. This is being stored separately from Users so we can run independent stats gathering without needing to access any actual user information. Because of the advantages of efficient JSON storage, we want to store the entire Collection object so we can retrieve the entire contents of a collection in a single query without requiring any joins.

We are storing CollectionItem as a separate item so that we can do future analysis on user-editable fields. We will not, however, store the full Game object under CollectionItem and simply store a link to the gameID so that we can ensure updates to Game are reflected when users view one of their CollectionItems.
All fields where a user can override the core Game statistics should be included in CollectionItem. Currently the only known value is `gameWeight`, an incredibly subjective measurement. Others may reveal themselves as this service nears implementation. In order to ensure a truly RESTful API, each element should map to a single URL, so both the Collection and CollectionItems should support the full set of CRUD operations as follows:
GET – This is a read request driven by either `collectionID` or `itemID` respectively.
PUT – Update the corresponding object with provided content updates.
POST – Create a new collection or collection item.
DELETE – Delete a collection or collection item. While every user should have one and only one collection, users can and will remove items from their collection.

Chief parameters should consist of `/collection?collectionid=<collectionID>` and `/collection/item?itemid=<itemID>`.

This operation is critical to collection management. This is really how users will manage their collection. While this will probably fall into a later implementation stage, it is vital to the success of this application.

- **`/wishlist`**

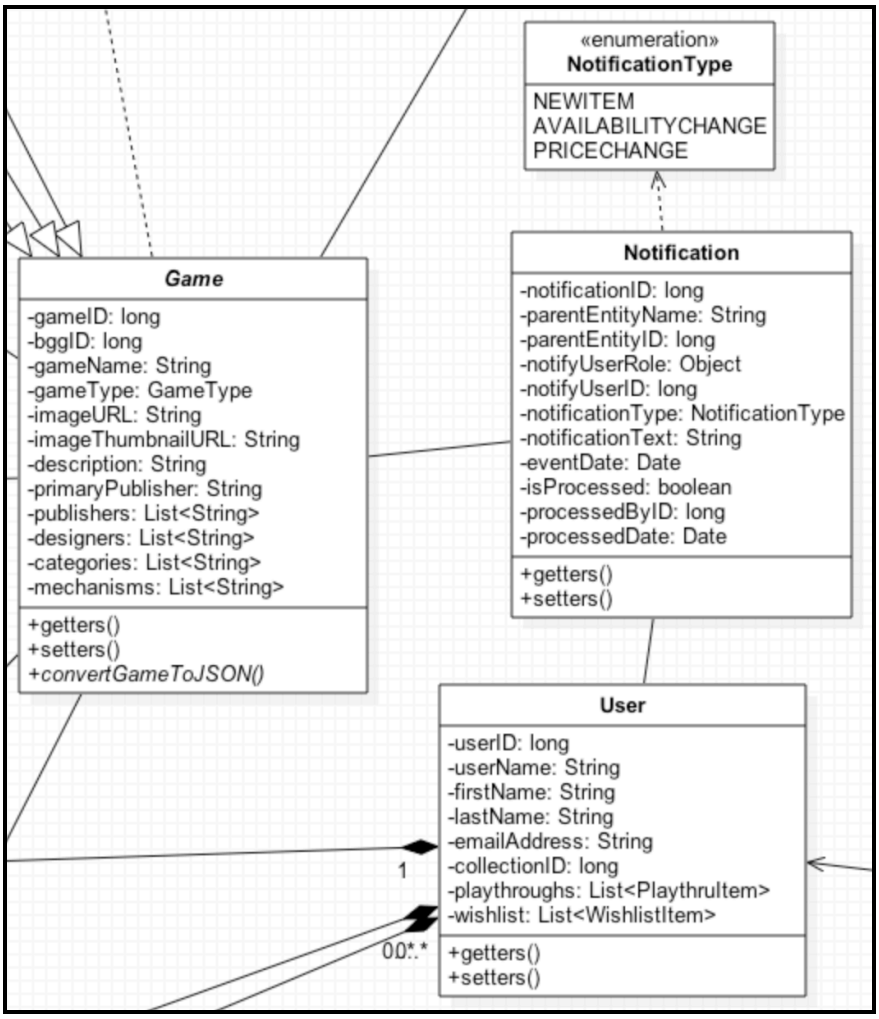


This is a pretty basic class. Users can have any number of items wishlisted. We are storing these independent of the User class (and inside the user class to allow for quick retrieval) so we can generate appropriate notifications without having to mine all User objects to figure the same information out. Whenever an item gets added, the corresponding User record will need to be updated to ensure that this content remains consistent. The following operations should be supported:
GET – This is a pretty basic read request by the `wishID`.
PUT – Update wishlist record with provided content updates. This should almost never happen. PUTs should trigger corresponding update requests for the User record.
POST – Create a new Wishlist item. Write the item and PUT it into the User's wishlist.
DELETE – Delete a WIshlist object from database by `wishID`. This could actually have two reasons. First, the user no longer wants this game, and is just getting rid of it. It could also be part of a transaction where the user acquired the game and is converting into a new CollectionItem. In either case, the User record will need to be updated as well.

Chief parameters should consist of `/wishlist?wishid=<wishID>`. Additional search operations may be added as we begin mining this data for content.

This operation is important, but definitely lower priority than many of the other tasks presented.

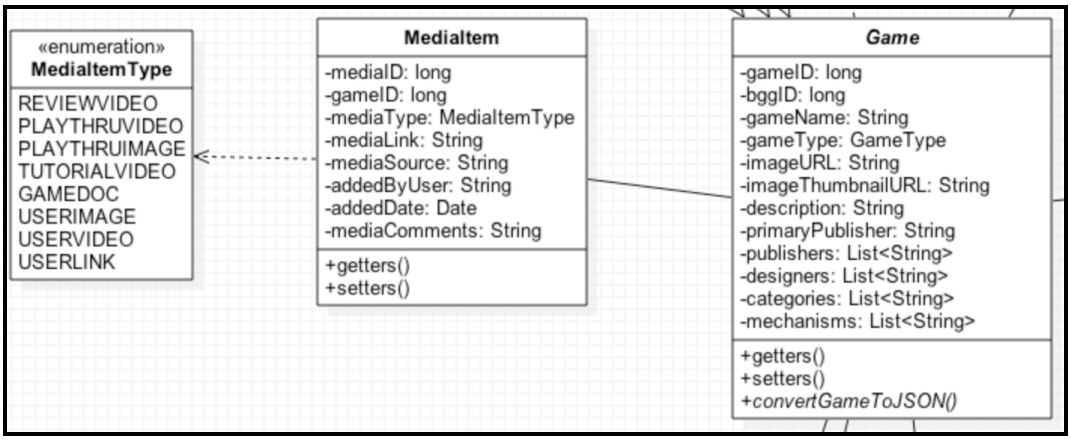
- **`/notification`**



This process is still in the process of being defined. The idea behind this idea is that the pull requests drive by the Admin users and background agent processes trigger important notifications for users, which for the time being will be stored in the database. This operation should support basic CRUD operations, though in practice, DELETE should not be used in an effort to build a history of all notifications processed by the system.

This operation is low priority, and will receive a more detailed exploration later in the project timeline.

- **`/mediaitem`**



This process will allow us to associate media content of various types to games. This might consist of review videos, game manuals, player aids, or other content. One unresolved question in this space is scoping. The idea behind storing media independent of the Game or the User is to allow for content sharing, but Users should have the option to keep some content private. Basic CRUD operations should be supported for this operation. Common GET requests might also use gameID or addedByUser as query parameters.

This approach is still being investigated and is considered low priority at this time.

Design of Mobile Client

The mobile client design process has not yet begun. At this point in the planning process, no real consideration has been given to visual presentation, which is appropriate for an agile approach. I have no experience with either web development or mobile development, so those are tasks that I have not yet begun to research. This work probably won't begin until at least the second project iteration cycle, if not the third.

There are two visual elements that will require the most design thought: How to represent a list of games cleanly, and how to display a single game and its associated content. How these tasks are done may vary greatly based on the amount of screen real estate the client has available.

My first approach will be to use a responsive layout design leveraging jQuery Mobile and Bootstrap. Depending on exposure and development time remaining, I may purse an Android app as the Mobile client. This will largely depend on project timelines and service implementations. Until I can become more familiar with these concepts, there is no firm design or technology commitment that could make that will have any value this early in the planning process.

V. Plan by Services

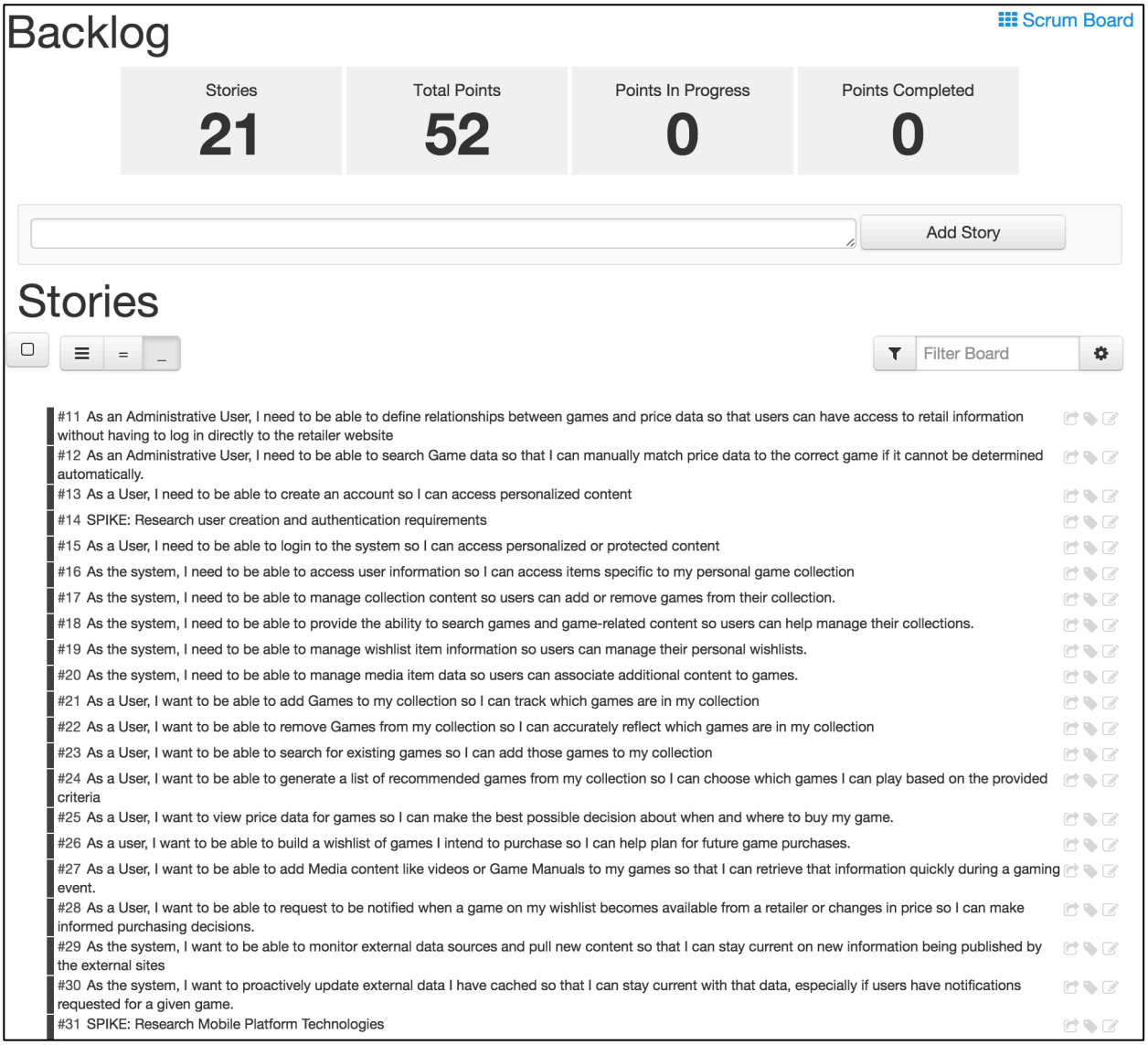
1. Project Scheduling in ScrumDo

There will be some differences in approach for this plan to those proposed in the Project Plan Guidelines. The first key difference is that this project will be managed in an Agile fashion, meaning that all new stories are added to and prioritized in the backlog and only the current iteration has stories assigned. This allows for the fact that the backlog story list may not be complete initially and will change over the course of project development.

This approach requires that at the end of each iteration, two things must happen. First, each iteration must be closed, with the completed stories resolved and closed and unfinished stories rolled into the next iteration. At this time, documentation should be revisited and updated, and all software artifacts should be in a stable state. Second, an iteration planning event must happen in which prioritized stories are added to the iteration from the backlog. During this event, overall progress on the project should be assessed to determine if sufficient progress was made, and priorities can be adjusted accordingly. This dynamic approach will allow for greater flexibility on this project, and more closely adheres to the traditional agile planning process.

This project is being tracked in ScrumDo under the "Terrapin Collection Manager" project [18]. At this moment, there are 31 stories planned for this project. More will probably be added as the project progresses. The first iteration, which will complete February 24th, has 8 stories currently assigned, and the following iteration has 2 stories assigned. 21 stories are currently roughly prioritized in the backlog.

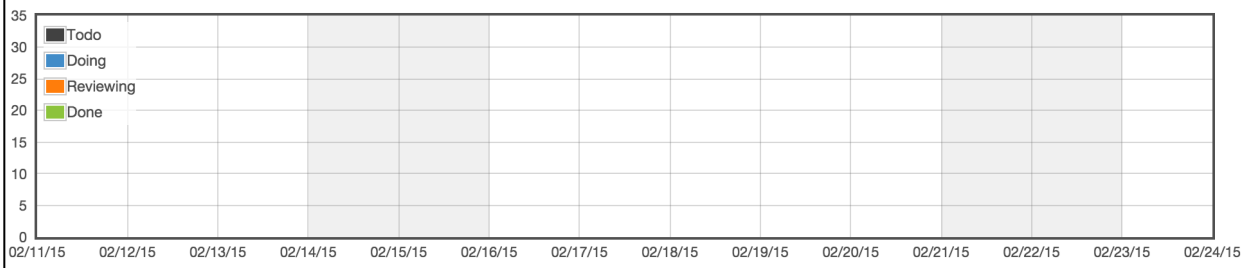
Screen shots are provided presenting the current content of each area:



First Iteration Feb 11, 2015 - Feb 24, 2015

Scrum Board

Stories	Total Points	Points In Progress	Points Completed	Days Left
8	31	3	0	13



Burnup | Burndown | Stacked | Time

Add Story

Stories

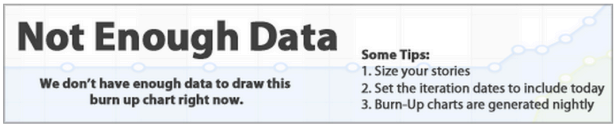
Filter Board

- #1 As the system, I need to be able to access game information from BoardGameGeek so I can build an index of game content.
- #2 As the system, I need to be able to access game and price information from coolstuffinc.com so I can build an index of game pricing information.
- #3 As the system, I need to be able to access game and price information from minaturemarket.com so I can build an index of game pricing information.
- #5 As the system, I need to be able to store external content in the database so that I can build an index of gaming and pricing content.
- #6 As an Administrative User, I need to be able to view new BoardGameGeek content so I can verify accuracy prior to conversion into Game content.
- #7 As an Administrative User, I need to be able to view new CoolStuffInc content so I can verify accuracy prior to conversion into Game content.
- #8 As an Administrative User, I need to be able to view new MiniatureMarket content so I can verify accuracy prior to conversion into Game content.
- #10 As an Administrative User, I need to be able to submit verified Game content so I can build the index that will drive the collection management tool.

Second Iteration Feb 25, 2015 - Mar 17, 2015

Scrum Board

Stories	Total Points	Points In Progress	Points Completed
2	18	0	0



Burnup | Burndown | Stacked | Time

Add Story

Stories

Filter Board

- #4 As the system, I need to be able to access game and price information from Amazon.com so I can build an index of game pricing information.
- #9 As an Administrative User, I need to be able to view new Amazon content so I can verify accuracy prior to conversion into Game content.

2. Project Management

Because I am functioning as a team of one, I will be serving as ScrumMaster, Product Owner, and all Team Members in the traditional Scrum model. Because of this fact, there will be less need for regular Scrum meetings, though project progress will be evaluated at regular points during an iteration to evaluate how I am progressing through the planned tasks.

The more valuable planning metric right now is project timelines. The following represent key dates and iterations have been planned to reflect this process:

Iterations

	Name	Stories	Start	End
	Backlog	21 stories		
	First Iteration	8 stories	Feb 11, 2015	Feb 24, 2015
	Second Iteration	2 stories	Feb 25, 2015	Mar 17, 2015
	Third Iteration	0 stories	Mar 18, 2015	Apr 07, 2015
	Fourth Iteration	0 stories	Apr 08, 2015	Apr 28, 2015
	Wrap-Up Iteration	0 stories	Apr 29, 2015	May 07, 2015

In terms of project milestones by product tier, my current plan involves the following:

- Completion of all initial web-service offerings by the end of the second iteration.
- Fully functional Administration client by the end of the second iteration.
- Basic functionality of User client by the end of the third iteration.
- Fourth iteration spent troubleshooting and cleaning up client-presentation elements.

VI. Risk Management

There are several key risks that exist with this project that warrant mitigation.

First, lack of familiarity with client-side technologies. This is a deficiency that is being remedied through labs and tutorials as part of this class. These proficiency gaps will also be addressed by additional spike stories as needed before approaching these tasks.

Second, with deployment options uncertain at this time, this leaves some uncertainty around what deployment options will exist. To mitigate this risk, web services will be designed to package as a WAR (java web application resource), since most service hosts support the war deployment strategy.

The final risk that will be mitigated during the project is the reality of working independently on this project while working fulltime and producing output comparable to the standard team size of four. This imbalance is offset some by my industry experience. To mitigate this risk, I will be actively managing my backlog and determining prioritization to best leverage my experience early in the project.

Technologically, there are additional risks that need to be considered. The first is security concerns relative to user creation and authentication. One potential means of mitigating this risk would be to rely on external security validation like OpenID [16,17]. This could result in changes to the User and UserDetail classes, but will need to be assessed once this project is ready to handle user validation.

VII. Bibliography

[1] – http://www.boardgamegeek.com
[2] - http://boardgamegeek.com/thread/1096301/2013-boardgamegeek-analytics
[3] - https://itunes.apple.com/us/app/boardgamegeek/id295454682?mt=8
[4] - https://play.google.com/store/apps/details?id=com.boardgamegeek
[5] - http://boardgamegeek.com/wiki/page/BGG_XML_API&redirectedfrom=XML_API#
[6] - http://boardgamemenu.com/
[7] - http://s3.amazonaws.com/WhatToPlay/download.html
[8] - https://www.behance.net/gallery/21540589/Hobby-Collection-django
[9] - http://sourceforge.net/projects/datacrow/?source=directory
[10] - http://sourceforge.net/projects/recipetools/?source=directory
[11] - http://sourceforge.net/projects/brickutils/?source=directory
[12] - https://play.google.com/store/apps/details?id=net.lp.collectionista
[13] - https://play.google.com/store/apps/details?id=com.spencerpages
[14] - https://play.google.com/store/apps/details?id=recinos.joel.bbcard
[15] - http://docs.aws.amazon.com/AWSECommerceService/latest/DG/ItemLookup.html
[16] - http://openid.net/add-openid/add-getting-started/
[17] - http://openid.net/add-openid/
[18] - https://www.scrumdo.com/projects/project/terrapin-collection-manager/summary