CARDS AGAINST HUMANITY

THE SERVER FOR THE ANTI-SOCIAL APP

System Documentation

BY ADAM ATRI
As part of THE ADAMS
CSC 289.0001 Summer 2014

TABLE OF CONTENTS

INTRODUCTION

The Cards Against Humanity Game Server provides web-based services for managing game play between multiple users of the Cards Against Humanity apps. It provides automated functionality for starting and maintaining multiple concurrent games, creating and verifying players, and persisting pertinent information about both. It also provides customized exceptions that assist in smooth game play, and it captures other exceptions writing them to an exception log. It is currently re-configurable only through the source code, however future iterations would include a configuration feature.

System Design

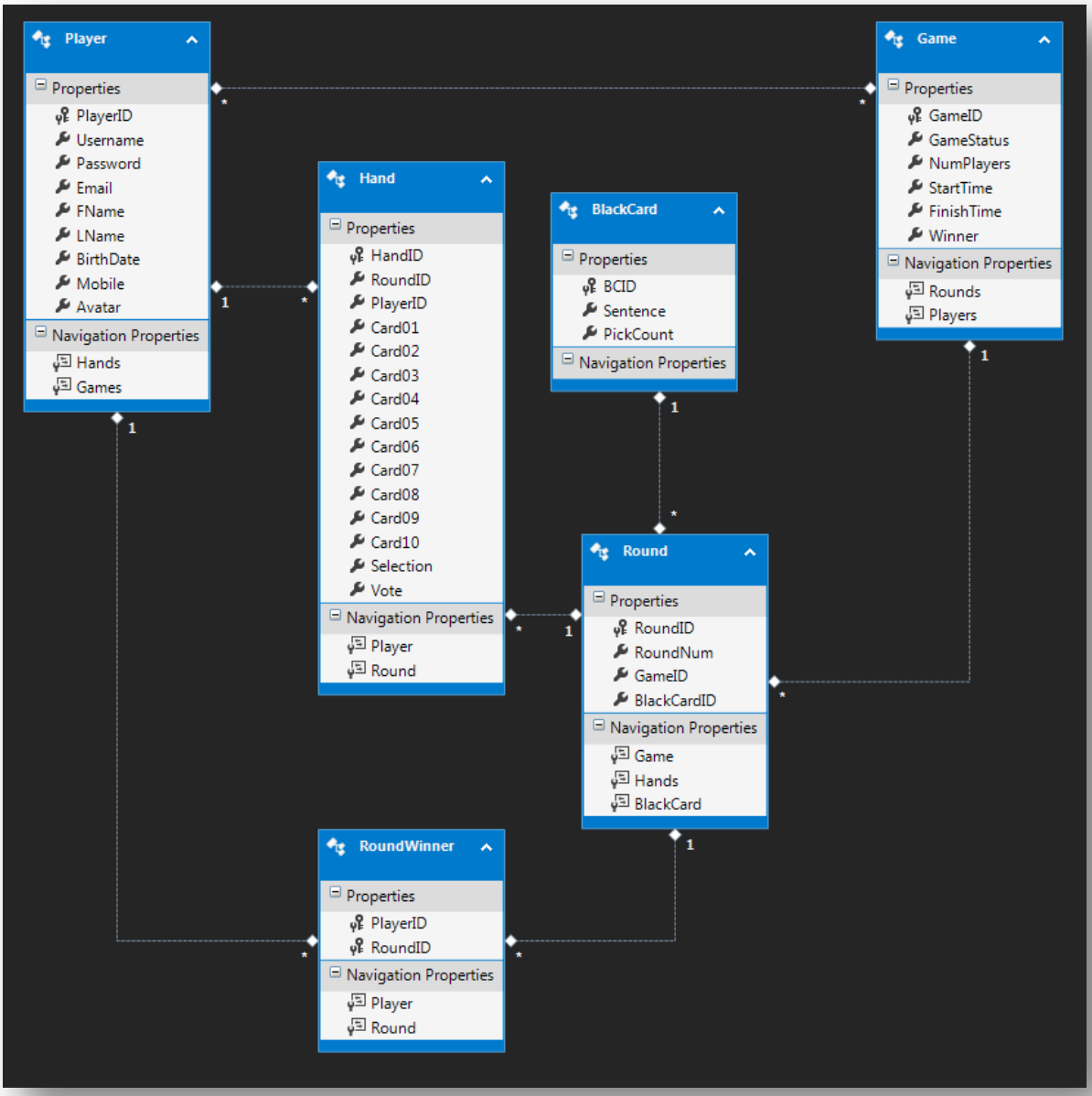
The project was intended to provide a web-accessible backbone for a multiplayer game application, and to retain information about those games and players to a database. To meet that goal, the server application, written entirely in VB.NET, is composed of three primary components: a Data Library that implements the Entity Framework, a WCF REST service, and a console application that hosts the service.

The whole project was an exercise in developing with new technologies by taking a fun and freely available concept and virtualizing it. Each of the technologies used (WCF, Entity Framework, LINQ, Task Parallel Library (TPL), JSON) are the currently accepted, and recommended, standards in .NET development. The use of these technologies was both challenging and rewarding, and further, it was sensible. These technologies, particularly the integration of the Entity Framework and LINQ, were meant to be used together and, while there was a considerable learning curve (facilitated by many online tutorials and message boards), led to a truly robust and highly extensible application.

The intended user for the application is a fairly limited audience, being useful only to those who would want to host the game, and who would also want have access to the data that the server retains. The extensibility of the data model may be useful to those that would want to capture more data than is currently being retained. Also, once initialized, there is very little that a user would be able to do, as the service is automated and not intended for extensive user interaction.

The CAH Data Library

The Data Library implements the Entity Framework in its own .NET class library. The entities, and their relations to one another, used throughout the program were designed here. It provides the context by which data is persisted and retrieved. It also generated the script that was used to build the database. The classes generated from the entity model were extracted to their own class library (CAH_EntityLibrary) so that they would exist in their own namespace (also a current standard for extending entity classes).



The data model as created through Visual Studio, using the Entity Framework

THE CAH GAME SERVICE

The service is the primary package in the project, containing all of the logic used to start, maintain, and finish games. It was implemented as a Windows Communication Foundation (WCF) REST service rather than as a SOAP service to facilitate platform independent client development. There are several components to the service that deserve mention including: The primary Game Classes, the ActiveGameTracker class, the Data Contract classes, the Data Repository class, the custom ExceptionsLibaray, and the ExceptionLogger (each of which will be further examined in the Implementation Section).

The main service interface, CAH_GameService >> ICAH_Service.vb, exposes all of the functionality that a client app would have access to. There are 9 exposed functions, exposed as URLs, that allow client apps to CreateNewPlayer, Login, JoinGame, GetHands, GetRoundStuff, SubmitSelection, CastVote, GetRoundWinner, and GetGameWinner. The CAH_Service implements the interface.

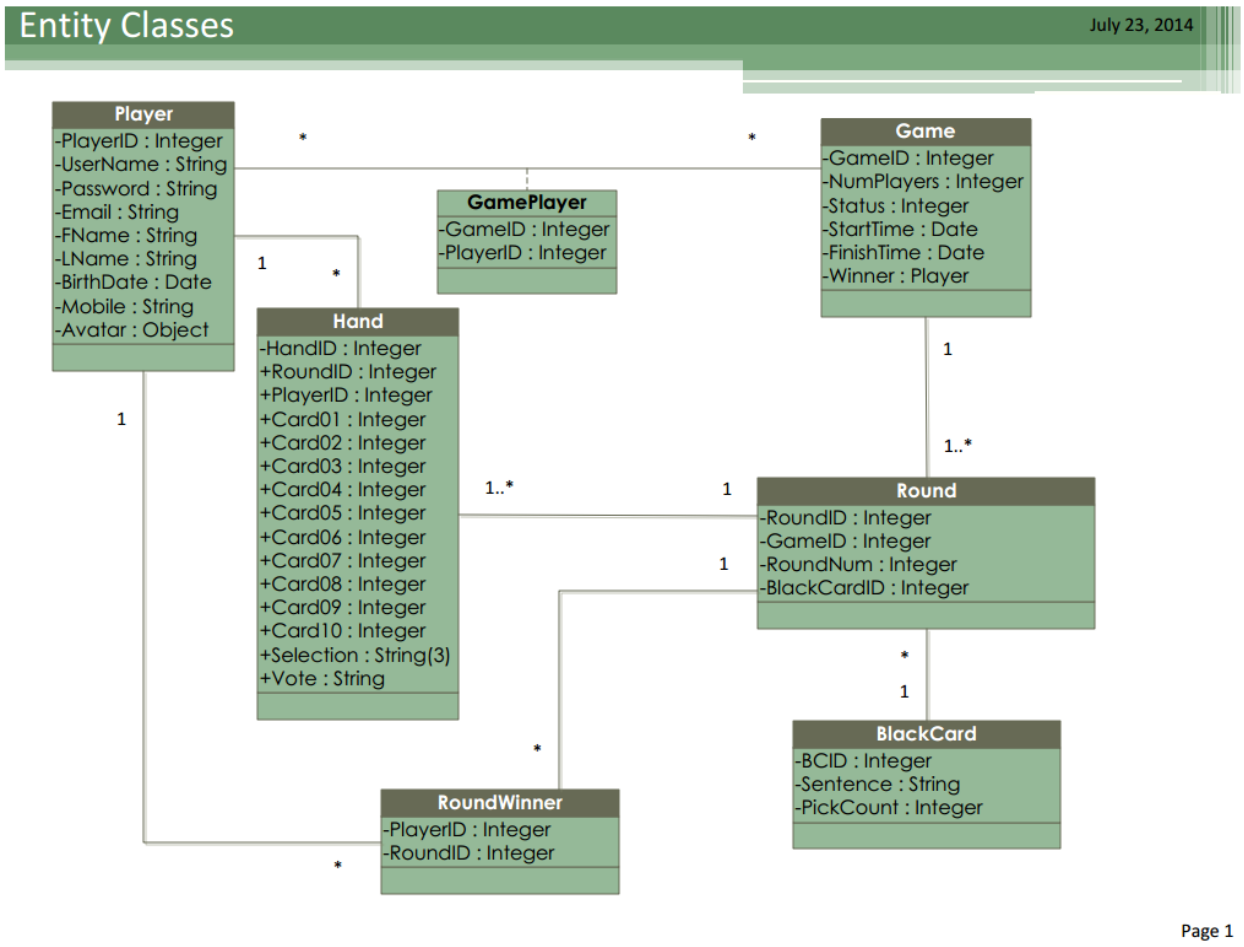
Outside of the .NET universe, coding for SOAP messaging can often be cumbersome and time-consuming. Since one of the intended goals of the service was the ability to server platform independent client applications, this was of primary. While some platforms do contain a SOAP API, all platforms support RESTful messaging. Also, SOAP requires all messages to be encoded in XML, which can be considerably more weighty than the JSON counterpart. As the number of games and players increases the time consumed to create and send messages could be minimized by using JSON. Each of these considerations directed me towards a RESTful implementation, which WCF was elegantly capable of providing.

THE CAH CONSOLE SERVER

The console server is a fairly simple program that hosts the service, exposing the service endpoints to the network. While the service could have been (and still can be) hosted as an ASP.NET web service or as a Windows Service, the choice to use a console application as the host was one of convenience: it's simply easier to implement and troubleshoot.

THE IMPLEMENTATION

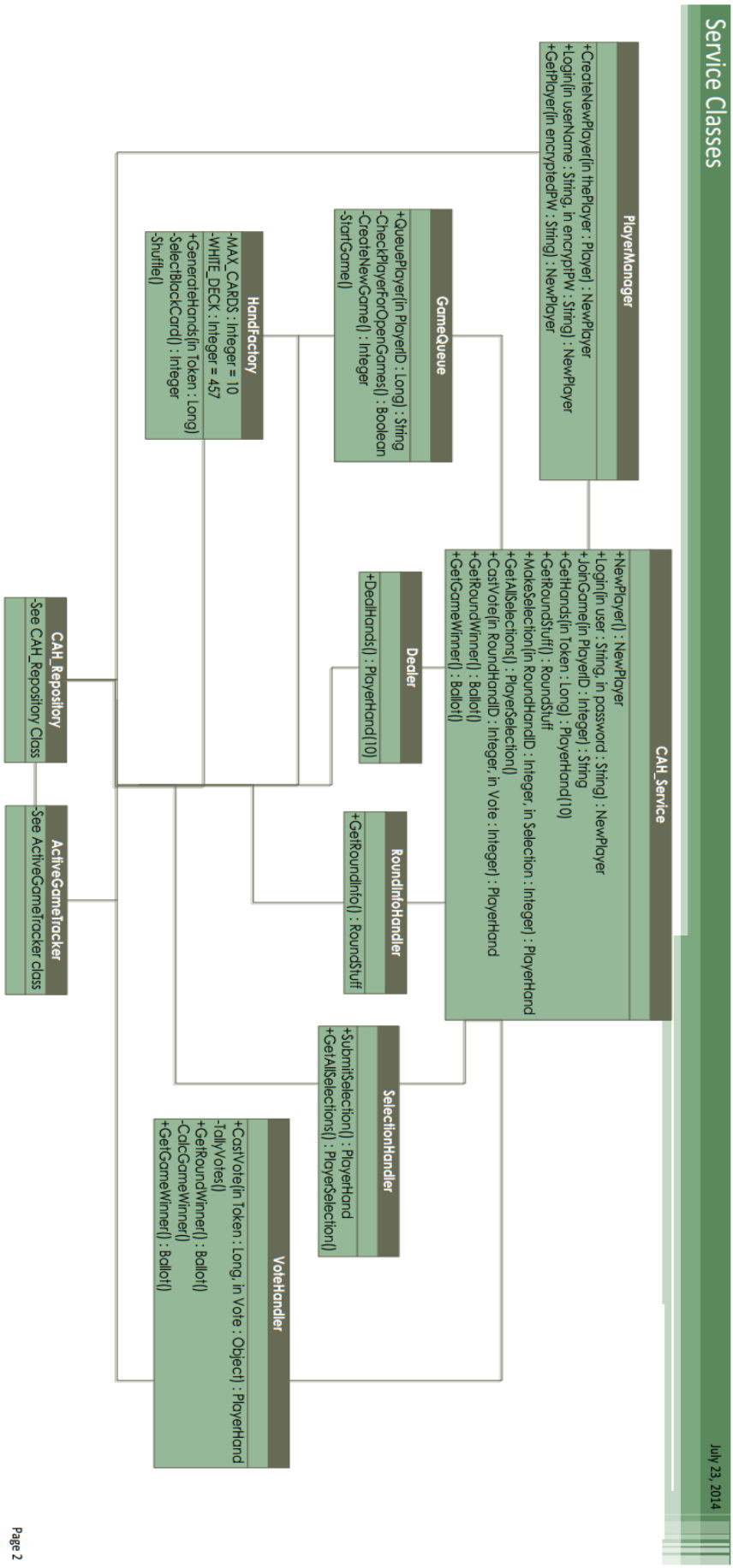
THE ENTITIES



The classes with the dark headers are the classes that are available through the CAH Data Repository. The GamePlayer table is a join table that simply expresses the Many-to-Many relationship between Games and Players. The Entity Framework creates the table in the generated SQL script, but does not expose this a useable table.

THE PRIMARY GAME (SERVICE) CLASSES

Each of the primary game classes provide specific functionality to facilitate game play.



PLAYERMANAGER

Allows clients to Create a new Player account and to login any time after creating an account. If the player attempts to create an account with a username that is already in use, the PlayerManager throws a *PlayerAlreadyExists* Exception. If a player attempts to invalid credentials, the PlayerManager throws an *InvalidLoginAttempt* Exception.

GAMEQUEUE

Is responsible for initializing games, adding players to the game, and telling the system to start the game when there are enough players. It also verifies that the player is not currently participating in another active game (only one game at a time). If the player is already involved with an active game, GameQueue returns the token necessary to rejoin the active game.

HANDFACTORY

When the GameQueue tells the system to start a particular game the HandFactory is notified to create all of the player Hands and all of the Round Information

DEALER

When the HandFactory has created the hands for a game, the players can request the hands from the Dealer. If the hands are not ready the Dealer throws a *HandsNotReady* Exception

ROUNDINFOHANDLER

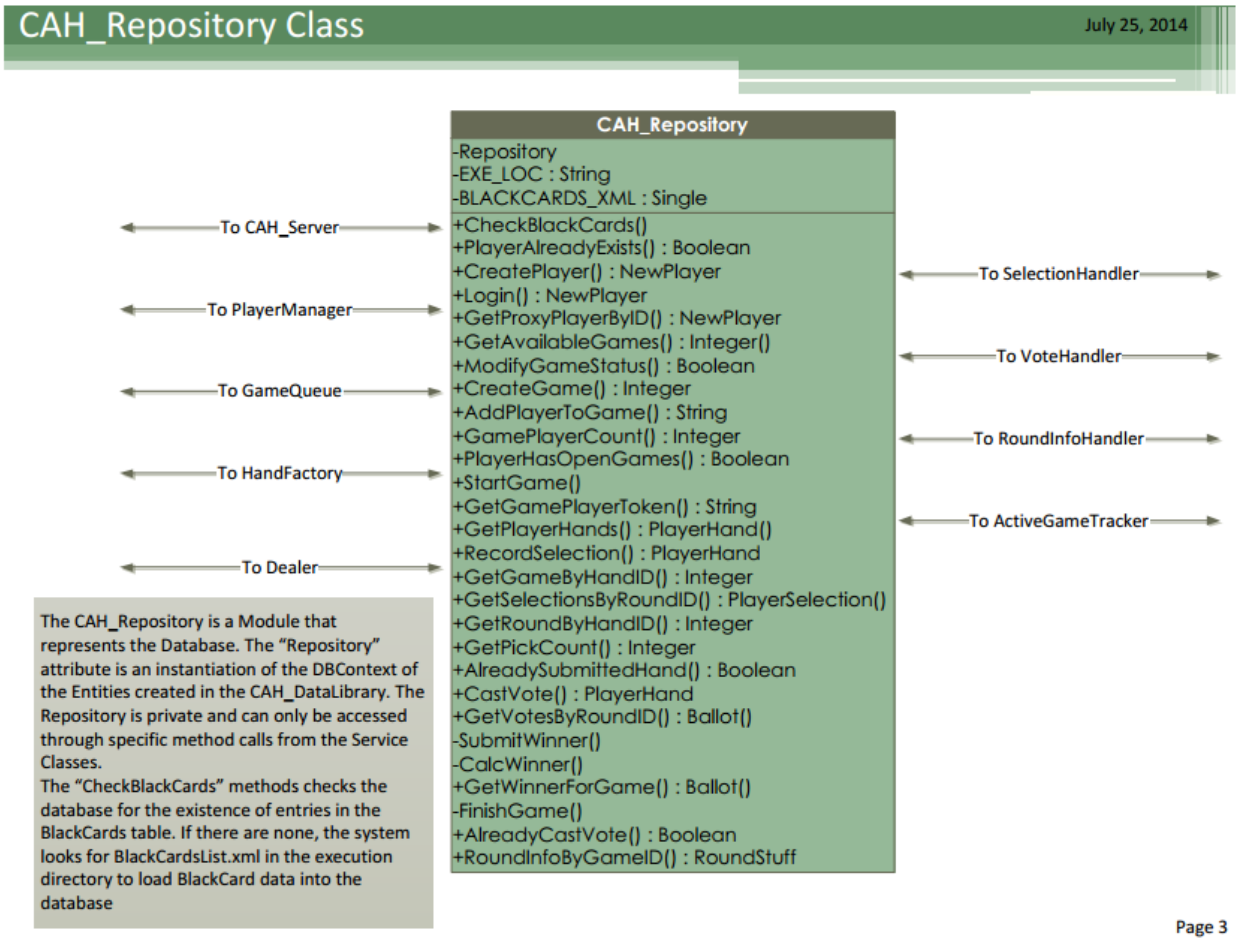
The RoundInfoHandler supplies players with information about their current round including the BlackCard and how many white cards to pick

SELECTIONHANDLER

Players have to make their best selection to fill in the blank. The SelectionHandler takes their individual selections and has the notifies the repository to save them. Once a player has submitted their selection, they will request all of the selections from all of the players, to vote on the best selection. If all players have not submitted their selection when an individual player tries to GetAllSelections, the SelectionHandler throws a *WaitingForAllSelections* Exception

VOTEHANDLER

Once a player has all the player selections for a round, the player will cast a vote for the best selection. VoteHandler accepts and records the vote. Once all votes have been cast, VoteHandler tallies the vote. When the vote has been tallied players can GetRoundWinner. If this is the last round of play, the VoteHandler also calculates the GameWinner. If a player attempts to GetRoundWinner before all the votes have been cast, the VoteHandler throws a *WaitingForAllVotes* Exception.



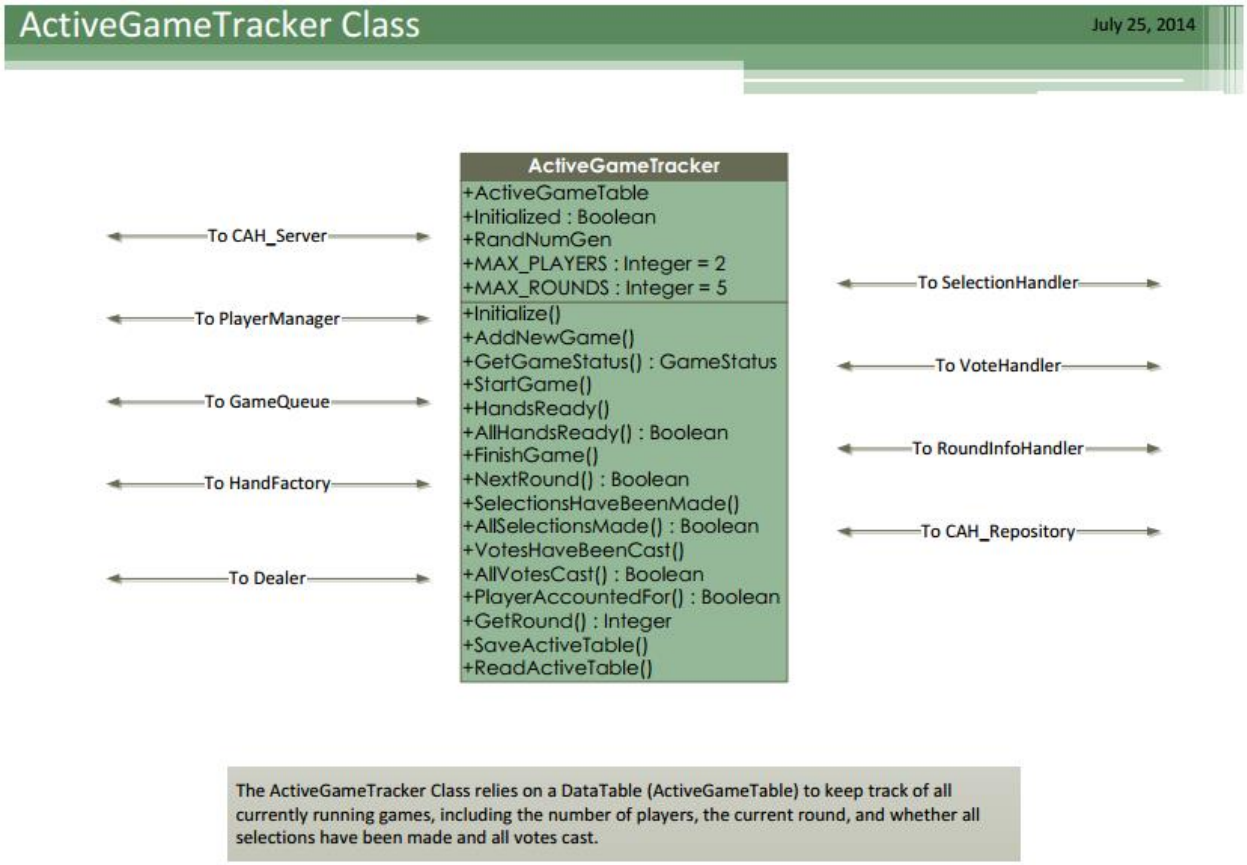
CAH_REPOSITORY

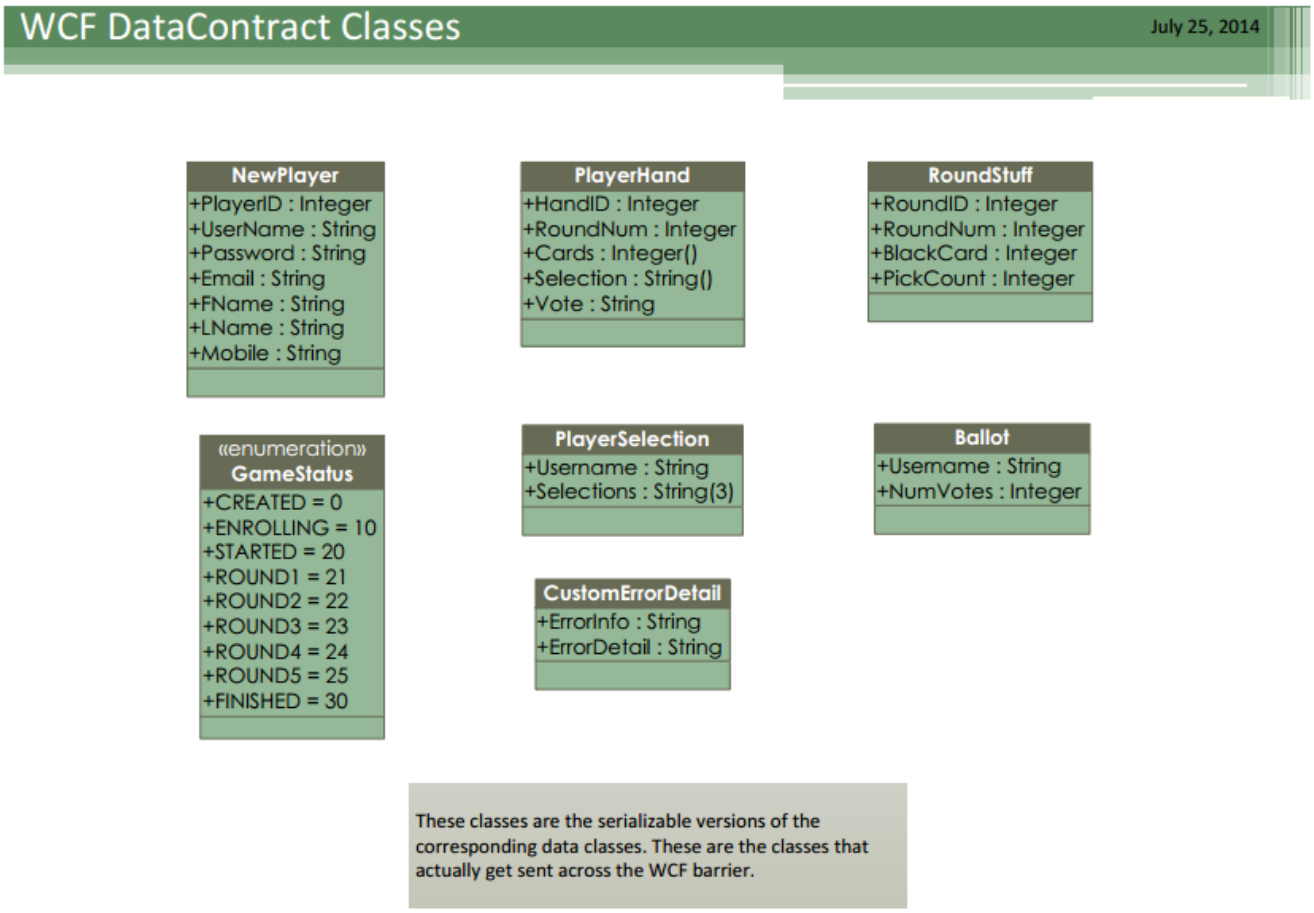
The CAH_Repository is a VB Module that represents the Database. The "Repository" attribute is an instantiation of the DBContext of the Entities created in the CAH_DataLibrary. The Repository is private and can only be accessed through specific method calls from the Service Classes.

The "CheckBlackCards" methods checks the database for the existence of entries in the BlackCards table. If there are none, the system looks for BlackCardsList.xml in the execution directory to load BlackCard data into the database

ACTIVEGAMETRACKER

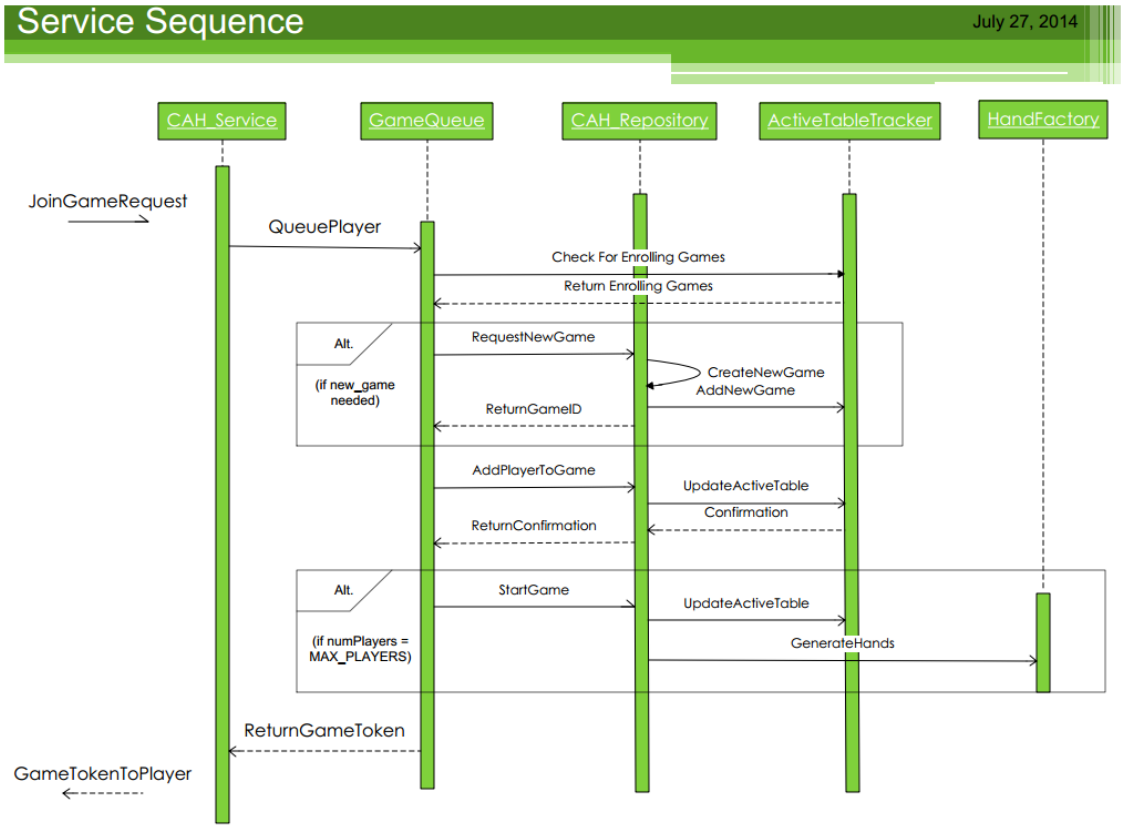
The ActiveGameTracker Class relies on a DataTable (ActiveGameTable) to keep track of all currently running games, including the number of players, the current round, and whether all selections have been made and all votes cast.

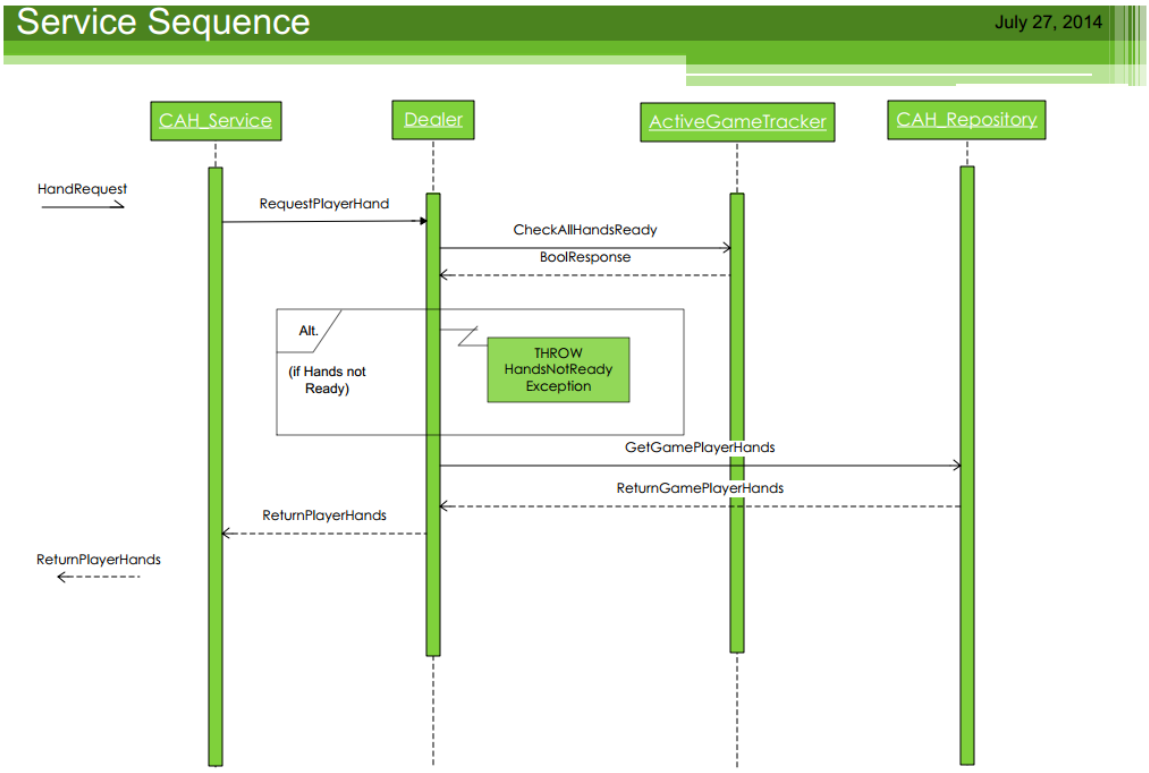


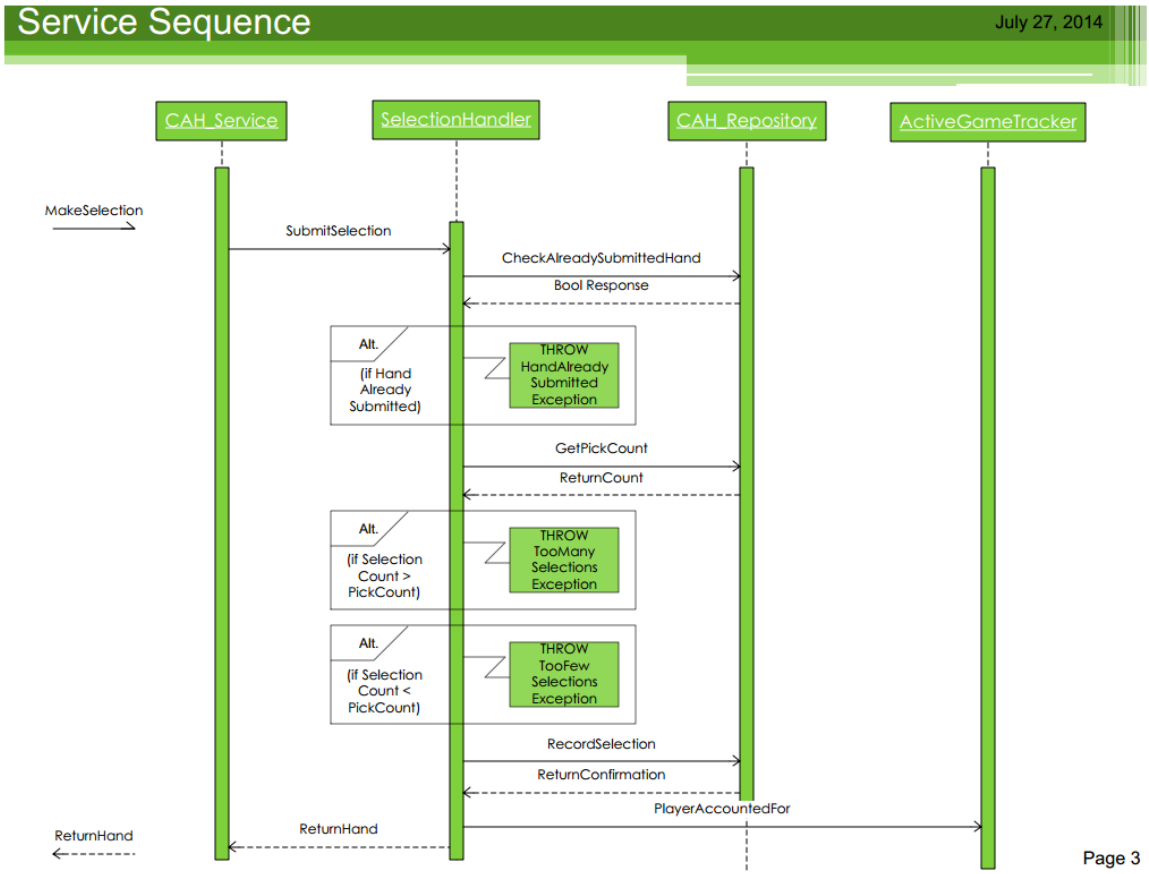


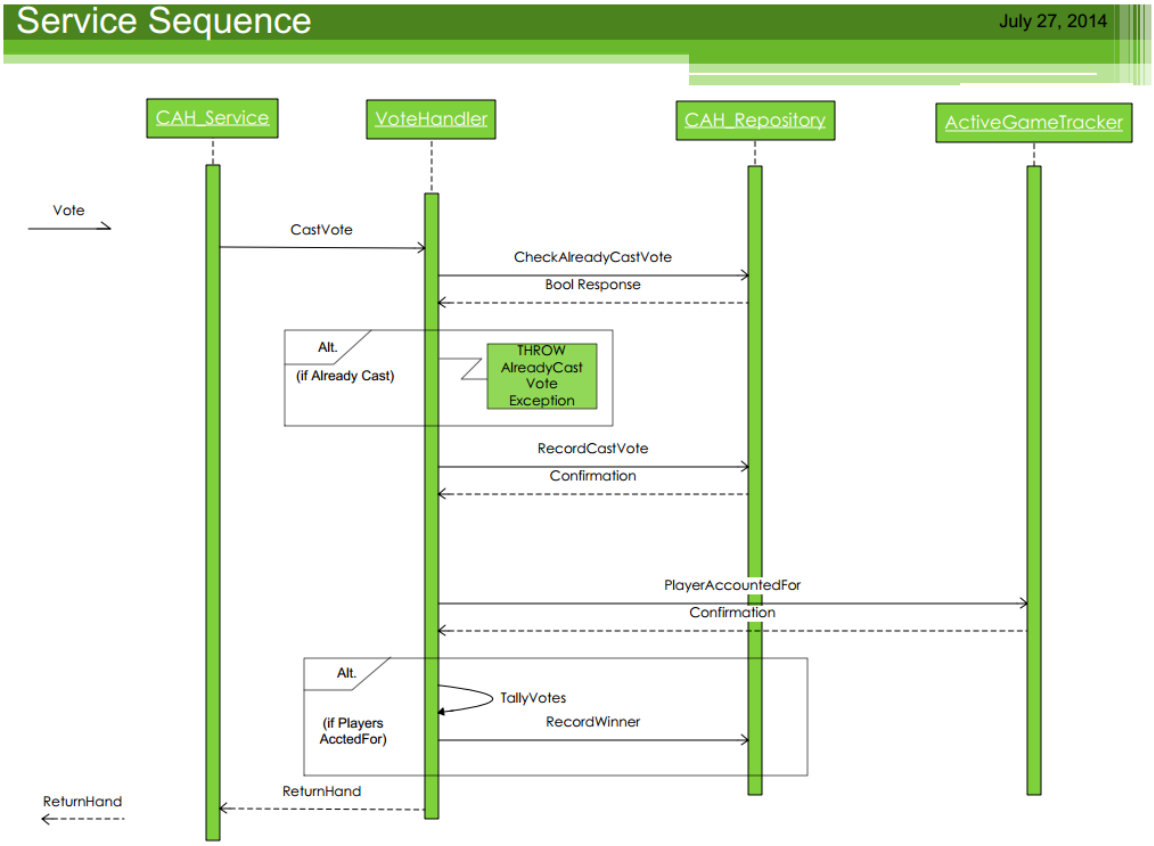
The WCF DataContract classes are the serializable versions of the corresponding data classes or parts thereof. These classes are converted to JSON objects and transmitted back and forth across the wire to the client application.

CAH SERVICE SEQUENCE DIAGRAMS






Service Sequence
July 27, 2014
CAH_Service
SelectionHandler
CAH_Repository
ActiveGameTracker
MakeSelection
SubmitSelection
CheckAlreadySubmittedHand
Bool Response
Alt.
(if Hand Already Submitted)
THROW HandAlready Submitted Exception
GetPickCount
ReturnCount
Alt.
(if Selection Count > PickCount)
THROW TooMany Selections Exception
Alt.
(if Selection Count < PickCount)
THROW TooFew Selections Exception
RecordSelection
ReturnConfirmation
PlayerAccountedFor
ReturnHand
ReturnHand
Page 3


Service Sequence
July 27, 2014
CAH_Service
VoteHandler
CAH_Repository
ActiveGameTracker
Vote
CastVote
CheckAlreadyCastVote
Bool Response
Alt.
(if Already Cast)
THROW AlreadyCast Vote Exception
RecordCastVote
Confirmation
PlayerAccountedFor
Confirmation
Alt.
(if Players ActtedFor)
TallyVotes
RecordWinner
ReturnHand
ReturnHand

System Testing

Due to the fact the project is a web service, testing primarily was handled by using a product called Fiddler by Telerik, which allows a user to send and receive HTTP messages. Fiddler was used to call each method in the CAH_Service in a live game play environment. Results were verified in the database. Additional testing was performed with an actual Android client (developed by Adam Ali Diarbakerli). Further Testing should be performed before putting the service into production, however preliminary testing has been positive. Regression testing was not performed on the maximum number of concurrent games, however multiple games have run successfully.

Conclusion

By all accounts, this project has resulted in a successfully implemented WCF REST service capable of monitoring and managing multiple concurrent gaming sessions. Proper planning, including the creation of Data Models, Entity Relationship Diagrams, and Sequence Diagrams, was an invaluable step in the successful creation of the developed service. The use of modern technologies, including Entity Framework, LINQ, TaskParallelLibrary, and JSON, proved challenging and effective.

In future iterations there should be some modifications and additions. Of primary concern, is the ability for a user to configure Service Constants including the maximum and minimum number of players, the maximum number of rounds per game, and the maximum and minimum number of white cards per hand.

REFERENCES AND ACKNOWLEDGEMENTS

There are too many short answers garnered from the internet to list them all individually. That said almost all answers to every challenge were found in a select number of locations.

General Programming, WCF, TPL Resources:

- Stack Overflow: http://stackoverflow.com/
- MSDN: http://msdn.microsoft.com/en-US/

Entity Framework Resources:

- Julie Lerman:
  - PluralSight.com – EntityFramework / Getting Started with Entity Framework 5
  - http://thedatafarm.com/blog/