

Application Development Report

C#/ASP.Net/Azure Pokédex

Phillip Stevens 100347238

Application Information

Application Purpose:

The Pokédex application was created in order for fans of the Pokémon universe to have a Pokédex, a database of all the Pokémon within the multi-million selling franchise by Nintendo and Game Freak, available to them on their computing device. The Pokédex's function within the game is to hold key information on Pokémon seen by the player, information such as the ID, name, elemental types and the description given about the Pokémon, effectively a portable database.

As mentioned above, the Pokédex contains a lot of useful information for the player, the most important being the types of the Pokémon, using this information the player can be aware of any advantages they can take in battle. Advantages such as using an electric move against a water Pokémon in order to deal a lot more damage in key fights.

By creating an application, users are able to look up and filter the Pokémon by type to quickly figure out weaknesses to a Pokémon. It's especially useful due to the Pokédex being off limits during battle meaning that users cannot look up Pokémon types when they need it most.

Installation Guide:

The Pokédex application is contained within the downloaded folder, by extracting the Pokedex.exe along with its accompanying files, the application can be ran by simply double clicking. The application will then automatically attempt to connect to the web server and pull back the information stored within the database in order to populate the program. Once it has connected once, it will store the data inside a local cache, which is lost once the program is closed, allowing for the program to be used even if the internet disconnects, after the initial connection.

Application Technology

Technology Breakdown:

The Pokédex application can be broken into three main components, Azure SQL Database/Server Hosting, Azure WebAPI Server Application written using C# and the Pokédex client written using C# and Winform.

Azure SQL Database and Server:

Using Microsoft Azure services allowed for a quick creation of an online database and server. From the beginning of the project this was clearly the best choice for the application as it is a well document service with built in tools in both the IDE (Integrated Development Environment) which was Visual Studio 2015 as well as tools provided via their online Portal.

One of the main benefits to Azure was that Visual Studio allows for a simple setup of an application that links to one of their hosted servers providing that you setup the user credentials to match the account that the SQL server is linked to, creating a server side application that would later be used in the development of the project.

A major disadvantage however to using Azure is that the service can at times be unpredictable in its behaviour. Despite having disabled XML serialization (a process that can be used to send data to the client application) the program on occasion still attempted to serialize data using XML instead of the JSON language which is what the de-serialization library embedded into the client uses, causing a long line of issues with the development.

One of the biggest flaws with Azure occurred near the end of the development cycle. Azure produced an obscure bug that destroyed the entire database and server client whilst in use causing the entire project, one day before it was due, to be completely broken and attempted to be fixed within the last day. This however is where one of the biggest benefits to using Azure came in, because of this obscure bug that destroyed all of the work, I had to contact the support teams and with it being owned and ran by Microsoft, there was a fast response in order to help me solve this issue and to see exactly what had happened with the project.

Once contacted via social media (as their direct contact services require a billing subscription to have access to and slower response times), I was contacted back almost immediately, leading to a discussion of the issue. Upon the service team hearing of my issue, they immediately requested more information including screenshots and created a forum post as well as passing on my query to the support team so they could try to assist me further in the issue.


Ask a question

Search related threads

Search forum questions

Quick access ▾

Asked by:

 25 Points
Top 30%

Dani Muszbek Microsoft (M...
Joined Jun 2015
Dani Muszbek Mic...
Show activity


Top related threads


- kb970895 broke my sql 2008 db client installation
- Windows update broke my Database?
- Need to use MID function in SQL
- Broke my user default database with .ChangeDatabase("master")
- Is Azure Elastic Scale is useful for mid-sized DB?


My SQL DB broke mid use.


Microsoft Azure > Azure SQL Database

Question

 From: Phil Stevens @PhiltheFaz via Twitter

 Hello I was wondering if there was anyone I could contact about my entire sql database breaking mid use?

 0 I was able to use my webAPI command of api/POKEMONs which placed all of the content of my database up on the web page during testing. It then said there was an error about serializing it to Json despite previously fixing that issue with some code which was still in place. It then just refused to display anything via the command and broke my client.

 Sign in to vote

Thanks

@AzureSupport

2 hours 53 minutes ago

Reply | Quote

Dani Muszbek Microsoft (MSFT CSG) 25 Points

Figure 1: Screenshot of my query being posted by the support staff at Azure Support (via: <https://social.msdn.microsoft.com/Forums/en-US/f1776abd-86c9-4ba9-81cd-18b7d8f0298c/my-sql-db-broke-mid-use?forum=ssdsgetstarted>)

Had my servers been hosted with Heroku, OpenShift or even Googles App Engine, it would have taken days before I heard anything back about the issue as the social media accounts for their support teams are a lot slower at responding and if the query was sent via email it would have taken several hours to days before a response would be received.

Azure WebAPI:

Azure's online tools didn't just allow for an easy time when creating the database, it also provided a simplistic way to begin the creation of a server that would act as a middleman between the client and the actual database. The web url is: <http://pokedexserver.azurewebsites.net/Help>. One of the biggest benefits to the of using Azure is that the database changes on the server have to be published via the Visual Studio suite which in turn requires a username (philthefez) and the password (Grandia1991) to push any changes made to the code.

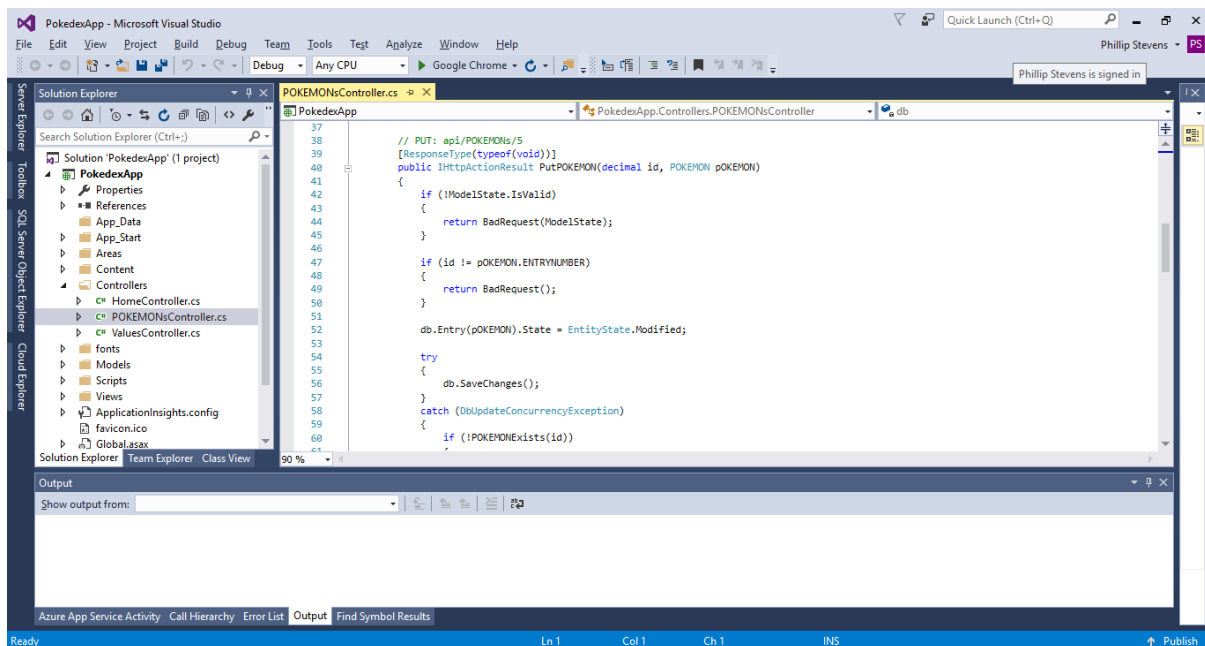


Figure 2: PokedexApp – An early look at the server side code project used as the middleman between the client and Azure Server

Once the database was setup it was easy to connect and debug the API tools to see if the server would be capable of handling the requests from the client program. This was done by publishing the changes made via this code project, to the server and then loading up the specified url along with the command. An example of the commands (which would later be used in the client) is api/POKEMONs which returned a list of all the Pokémon stored in the SQL database which would have been passed down by the server to the client in order to populate the clients list thus giving the program information to manipulate.

```

[{"id":1,"ELEMENT1":{"$ref":"2"},"POKEMONs":[{"$ref":"1"}],"ELEMENT2":{"$ref":"4"},"POKEMONs1":[{"$ref":"1"}],"POKEMONs2":[{"$ref":"3"}],
{"id":5,"ELEMENT1":{"$ref":"2"},"ELEMENT2":{"$ref":"4"},"GENERATION1":{"$id":"6"},"POKEMONs":[{"$ref":"1"}],"POKEMONs1":[{"$ref":"3"}],
{"$ref":"7"}],"POKEMONs1":[{"$ref":"7"}],"ELEMENT1":{"$id":"8"},"POKEMONs1":[{"$ref":"6"},"ENTRYNUMBER":22.0,"NAME":"CHARMANDER",
"DESCRIPTION":"Obviously prefers hot places. When it rains, steam is said to spout from the tip of its tail.",
"GENERATION1":{"$ref":"1"},"GENERATION2":{"$ref":"1"},"ENTRYNUMBER":21.0,"NAME":"VENUSAUR",
"DESCRIPTION":"The plant blooms when it is absorbing solar energy. It stays on the move to seek sunlight.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":20.0,"NAME":"IVYSAUR",
"DESCRIPTION":"When the bulb on its back grows large, it appears to lose the ability to stand on its hind legs.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":19.0,"NAME":"BULBASAU",
"DESCRIPTION":"A strange seed was planted on its back at birth. The plant sprouts and grows with this POKEMON.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":18.0,"NAME":"SANDSLASH",
"DESCRIPTION":"It has a habit of staying in its den for a long time. It appears to be a very lazy Pokémon.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":17.0,"NAME":"MACHOP",
"DESCRIPTION":"It has superhuman strength and is known for its feats of strength. It has a habit of staying in its den for a long time. It appears to be a very lazy Pokémon.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":16.0,"NAME":"MACHOP",
"DESCRIPTION":"It has superhuman strength and is known for its feats of strength. It has a habit of staying in its den for a long time. It appears to be a very lazy Pokémon.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":15.0,"NAME":"MACHOP",
"DESCRIPTION":"It has superhuman strength and is known for its feats of strength. It has a habit of staying in its den for a long time. It appears to be a very lazy Pokémon.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":14.0,"NAME":"MACHOP",
"DESCRIPTION":"It has superhuman strength and is known for its feats of strength. It has a habit of staying in its den for a long time. It appears to be a very lazy Pokémon.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":13.0,"NAME":"MACHOP",
"DESCRIPTION":"It has superhuman strength and is known for its feats of strength. It has a habit of staying in its den for a long time. It appears to be a very lazy Pokémon.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":12.0,"NAME":"MACHOP",
"DESCRIPTION":"It has superhuman strength and is known for its feats of strength. It has a habit of staying in its den for a long time. It appears to be a very lazy Pokémon.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":11.0,"NAME":"MACHOP",
"DESCRIPTION":"It has superhuman strength and is known for its feats of strength. It has a habit of staying in its den for a long time. It appears to be a very lazy Pokémon.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":10.0,"NAME":"MACHOP",
"DESCRIPTION":"It has superhuman strength and is known for its feats of strength. It has a habit of staying in its den for a long time. It appears to be a very lazy Pokémon.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":9.0,"NAME":"MACHOP",
"DESCRIPTION":"It has superhuman strength and is known for its feats of strength. It has a habit of staying in its den for a long time. It appears to be a very lazy Pokémon.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":8.0,"NAME":"MACHOP",
"DESCRIPTION":"It has superhuman strength and is known for its feats of strength. It has a habit of staying in its den for a long time. It appears to be a very lazy Pokémon.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":7.0,"NAME":"MACHOP",
"DESCRIPTION":"It has superhuman strength and is known for its feats of strength. It has a habit of staying in its den for a long time. It appears to be a very lazy Pokémon.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":6.0,"NAME":"MACHOP",
"DESCRIPTION":"It has superhuman strength and is known for its feats of strength. It has a habit of staying in its den for a long time. It appears to be a very lazy Pokémon.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":5.0,"NAME":"MACHOP",
"DESCRIPTION":"It has superhuman strength and is known for its feats of strength. It has a habit of staying in its den for a long time. It appears to be a very lazy Pokémon.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":4.0,"NAME":"MACHOP",
"DESCRIPTION":"It has superhuman strength and is known for its feats of strength. It has a habit of staying in its den for a long time. It appears to be a very lazy Pokémon.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":3.0,"NAME":"MACHOP",
"DESCRIPTION":"It has superhuman strength and is known for its feats of strength. It has a habit of staying in its den for a long time. It appears to be a very lazy Pokémon.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":2.0,"NAME":"MACHOP",
"DESCRIPTION":"It has superhuman strength and is known for its feats of strength. It has a habit of staying in its den for a long time. It appears to be a very lazy Pokémon.",
"GENERATION1":{"$ref":"6"},"ENTRYNUMBER":1.0,"NAME":"MACHOP",
"DESCRIPTION":"It has superhuman strength and is known for its feats of strength. It has a habit of staying in its den for a long time. It appears to be a very lazy Pokémon."}]

```

Figure 3: Example database output from early stages of the applications development, using the api/POKEMONs command.

Figure 3 shows an early example of what would be passed over via a message, if requested by the client program, which the client would then decode and use to populate its application.

API command creation is one of the strongest benefits to using the Azure system, as the tools built into the ASP.NET framework allowed me to create customised commands that could be used to return specific data be it individual items or even all of the items currently stored within the database.

Pokédex Client C# Winform:

The Pokédex client was created simply by using the in-built tools to Visual Studio, namely the C# Windows Form creation tools which easily allowed for the creation of interactivity and the design of the form as can be seen in figure 4 below.

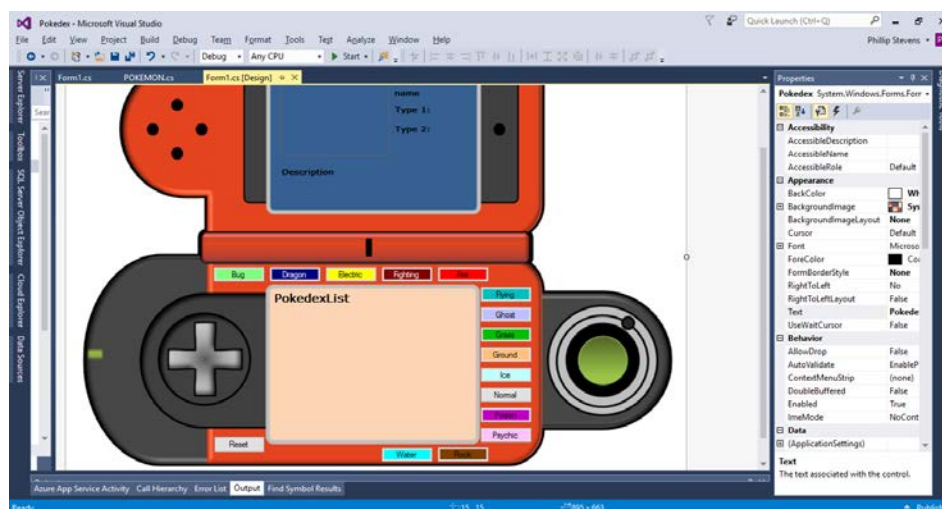
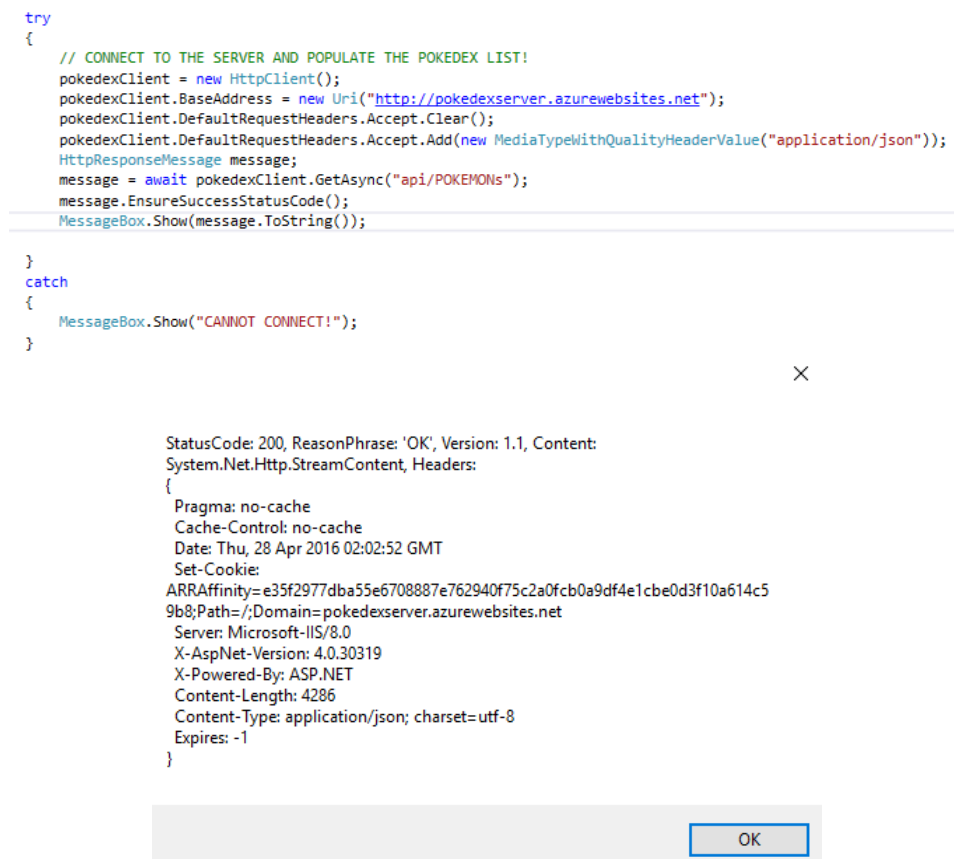


Figure 4: C# Winform Design Tools

As seen in Figure 4, there are numerous buttons placed on top of a picture creating the form that the user will interact with. By using the visual toolset it was easy to create buttons and then define their actions based on specified input including anything from a single click,

double click or even if the mouse simply moved inside the button. The visual toolset allowed for the exact creation of a user form without the constant need to debug the project and is something other services wouldn't have been able to provide.

Another one of the main benefits to using Visual Studios and the Windows Form tools is that the program contains libraries that can be accessed in order to begin querying any web API that I had access to. These tools were the main way that the client would contact the server in order to pull the information down. An early example of testing if the connection was made can be seen in Figures 5 and 6 below:



Figures 5 & 6: Early stage connection testing of WebAPI testing.

Developer Diary

Developer Diary:

Date	Work Time	Description
07 April	11am-11.30am	Received feedback from Wayne on my idea, got the go ahead and began to plan the project on paper by listing the assets and tools required.
08 April	11am-1pm	Finalized my list of tools required for the project: <ul style="list-style-type: none">• Azure Hosting• DreamSpark Account• Visual Studio 2015• Newtonsoft JSON De-Serializer Finalized my art asset locations: <ul style="list-style-type: none">• Pokedex image for the windows form• (unused) Individual Pokemon gif files• (unused) collection of Pokemon Types
09 April	11am-2pm	Created my DreamSpark account after having it reset months ago in order to attempt to download the latest Visual Studio and sign up for my Azure free 1 month Developer Trial. (Issue) Attempted to sign up to DreamSpark using my unimail account which required significant amounts of attention due to an unknown authentication issue on DreamSpark's end. (Fix) Finalized sign up to DreamSpark upon retrieving confirmation email around 2pm.
12 April	1pm-3pm	Now that the DreamSpark account was in place, I was able to sign up to my free trial for Azure and begin to develop the database. This became problematic due to long wait times for solving the issue. (Issue) The Azure website took a long time to respond and react to my trial account and allow me access through the web portal. (Fix) Contacted the Azure team to request that they look into the issue and responded almost immediately solving the issue within an hour.

15 April	2pm-5pm	<p>Began setting up the initial server and SQL database via Azure in order to begin populating it. Whilst waiting on the database and server to be setup I wrote the entire SQL database using Notepad++, including the layout of the relational database and some test data.</p> <p>(Issue) Azure sometimes have a small wait time when setting up databases and server and so created a long wait time.</p>
19 April	10am-3pm	<p>I initially ran the SQL query onto the SQL database using Azure's online tools and fixed a few small SQL inconsistencies such as spelling mistakes and incorrectly defining foreign keys within the code. Began work on the Server Side implementation in order to prepare the API that will be used in conjunction with the client program.</p> <p>(Issue) Upon the database being updated an error occurred when it came to viewing the data on the server application as it wasn't allowing me to create a view or a controller within the Server side application with the updated information.</p>
20 th April	1pm-5pm	<p>Upon fixing the previous days issue, I set out to create the model, controllers and views of the main table from within the relational database. Once done I tested the API by publishing it to the web and looking in the help section. It was here that I found that due to the foreign key setup, data was being processed and passed over that wouldn't need to be passed over, leading me to research setting up custom hash table information.</p> <p>(Fix) After resetting my credentials within Visual Studio, it allowed for the creation of models, views and controllers.</p>
22 nd April	10am-12pm	<p>After publishing all of the changes made on the previous day, I have begun the development of the client side application. I have mainly focussed on the creation of the applications design as that was a big part to the application as I didn't want to create just another windows form with little changed to make it feel unique.</p> <p>In order to achieve this I applied a picture to the form (of a Pokedex seen in the TV show) and removed the border of the window.</p> <p>(Issue) Borderless Window Mode has no handle to drag the program by so it is impossible for me to move the window.</p>
23 rd April	2pm-3.30pm	<p>I managed to fix the Borderless Window Mode issue from yesterday by creating a really simple method that allowed</p>

24 th April		<p>the window to move around the desktop, however this created an issue of its own. Whilst I didn't know how to fix this I setup the rest of the forms features including the buttons, picture boxes, labels and the creation of the events I needed to occur such as a button reacting to a single click.</p> <p>(Fix & Issue) Made the window moveable but it would always ignore where I clicked and just moved the mouse to the top left (origin point of the window) and dragged it from there creating a horrible drag animation.</p>
	12pm-4pm	<p>I began to link the client to the WEBAPI that was published and hosted online. It was a relatively new concept for me so I spent a lot of my time researching how to connect to a website and how to send/receive messages from it in order to make sure that my WEBAPI was reachable.</p> <p>In the end I managed to connect the client to the server and send it a message to see if it had connected, on return it threw a bug at me and development stopped when I couldn't fix the issues.</p> <p>(Issue) Connecting to the WEBAPI was fine however there is an odd bug that occurs when retrieving the message to check the contents of the message and placing it into a message box.</p>
25 th April	4pm-7pm	<p>Fixed the issue with the message box contents with relative ease, so I began to attempt to issue commands out to the WEBAPI, I did this by sending a message to the API set to a media type of JSON in order to retrieve the JSON serialized information that my API would pass over. Realized that C# didn't have a JSON de-serializer in place and had to research my options.</p> <p>(Fix) Used the .ToString() method to produce content that could be passed into the message box and displayed on screen.</p> <p>(Issue) Needed to locate and install a JSON deserializing library into C# in order to correctly obtain all of the data from the server side.</p>
26 April	11am-8pm	<p>Installed the JSON Deserializer called Newtonsoft which was available via the NuGet Packages installer built in to Visual Studio. Spent a lot of the time gaining and understanding on how to use it and eventually returned the message before promptly deserializing the data.</p>

		<p>From here I poured all of the information into a list and then used that list to populate both the local cache and the web cache (both of which are lists respectively). Once this worked I set out to populate the list box located on the main form which displays the intractable items from the database.</p> <p>Then from there I worked on the update method which will be called before every action in order to make sure the current information from the API was up to date.</p> <p>(Fix) Fixed the lack of JSON deserialization</p>
27 th April	11am-6pm	<p>Now that all of the information was in place I created all of the interactivity between the buttons allowing users to filter the Pokemon by their types.</p> <p>Once this was complete all of the functionality was in place and only required some careful debugging in order to fix unhandled exception errors when trying to select an index that wasn't populated.</p> <p>I later discovered there was a minor issue within one of my controllers and fixed the issue, however upon publishing the Server side application an issue occurred and it deleted my controllers. After some research I was able to find out that I could recall files from the current server and did so, only to find that it was corrupted and lost caused me to lose all functionality. Not only this but my SQL database had become corrupted and all data was lost.</p> <p>I promptly contacted Azure services in order to attempt having the issue resolved and they set me up with contacts via their social media accounts, posting on their forum on my behalf and setting up a contact point so I can contact them for further information without the billing issue.</p> <p>(ISSUE!!!) Entire database was corrupted and I lost all my controllers including my hash tables.</p>
28 th April	11am-8pm	<p>After contact with both Wayne and Azure, I attempted to salvage my data and repopulated my database by dropping all the tables and re-running my SQL queries that I had written at the beginning. This returned some functionality however due to the loss of the Hash Table setup, the data pulled by the client is limited to that of ONLY the first entry of the database.</p> <p>This severely limited the usage of the application however I</p>

placed in some test data in case of an issue arising during the demo which is only triggered if there is a failed connection to the API. Upon restoring only the first entry of the database I was able to use all of my functionality again within the form limited to ONLY that one entry.

Gathered proof of all my contact and placed it within the report.