

GALWAY-MAYO INSTITUTE OF TECHNOLOGY

**BrainWorks - A Brain Training App and accompanying Website**

***Authors***

**Alanna Curran**

**Claire Finn**

**Declan Duffy**

**Thomas McNamara**

***Supervisors***

**Brian McGinley**

**Martin Hynes**

***Academic Year***

**2016 – 2017**

***Submitted Date***

**Monday 17th April 2017**

***Submitted in support of the degree of***

**Bsc (Hons) of Science in Computing in Software Development**

Table of Contents

Contents

[ACKNOWLEDGEMENTS 4](#_Toc480137897)

[INTRODUCTION 5](#_Toc480137898)

[Running the application: 11](#_Toc480137899)

[METHODOLOGY 12](#_Toc480137900)

[TECHNOLOGY REVIEW 22](#_Toc480137901)

[Platforms: 22](#_Toc480137902)

[IDEs: 22](#_Toc480137903)

[Databases: 22](#_Toc480137904)

[Servers: 23](#_Toc480137905)

[Testing: 23](#_Toc480137906)

[Other Technologies: 23](#_Toc480137907)

[Languages: 24](#_Toc480137908)

[Universal Windows Application: 24](#_Toc480137909)

[Website: 32](#_Toc480137910)

[SYSTEM DESIGN 42](#_Toc480137911)

[Website UI: 50](#_Toc480137912)

[Cloud Hosting: 51](#_Toc480137913)

[Website Hosting: 51](#_Toc480137914)

[SYSTEM EVALUTATION 55](#_Toc480137915)

[Brainworks Website: 56](#_Toc480137916)

[SYSTEM EVALUTATION CONTINUED 61](#_Toc480137917)

[CONCLUSION 66](#_Toc480137918)

[LEARNING OUTCOMES: 68](#_Toc480137919)

[FUTURE INVESTIGATION: 69](#_Toc480137920)

[REFERENCES: 72](#_Toc480137921)

Illustrations

[Figure 1 - Universal Windows Application 10](#_Toc480137922)

[Figure 2 - Agile Development 16](#_Toc480137923)

[Figure 3 - Git Bash 20](#_Toc480137924)

[Figure 4 - GitHub 21](#_Toc480137925)

[Figure 5 - Facebook Developer 32](#_Toc480137926)

[Figure 6 - Valid OAuth redirect URI 32](#_Toc480137927)

[Figure 7 - Facebook Authentication 34](#_Toc480137928)

[Figure 8 - Login Successful 34](#_Toc480137929)

[Figure 9 - Internal Page 35](#_Toc480137930)

[Figure 10 - Example Game Synopsis 37](#_Toc480137931)

[Figure 11 – Swagger UI 39](#_Toc480137932)

[Figure 12 - Search score using Swagger UI 40](#_Toc480137933)

[Figure 13 - Displaying info from DB from Swagger UI 40](#_Toc480137934)

[Figure 14 - Sample HTML 42](#_Toc480137935)

[Figure 15 - Struct in Golang 42](#_Toc480137936)

[Figure 16 - Structure of Table on Azure 42](#_Toc480137937)

[Figure 17 - BrainWorks Games Scores on the UWP Side 43](#_Toc480137938)

[Figure 18 - BrainWorks Games Scores on the Golang Website Side 44](#_Toc480137939)

[Figure 19 - Golang code to access the scores 44](#_Toc480137940)

[Figure 20 - Application Main Page 50](#_Toc480137941)

[Figure 21 - Attention Page 50](#_Toc480137942)

[Figure 22 - Problem Solving Page 52](#_Toc480137943)

[Figure 23 - Speed Game's Page 52](#_Toc480137944)

[Figure 24 - Project Architecture 57](#_Toc480137945)

[Figure 25 - Selenium Test Case 58](#_Toc480137946)

[Figure 26 - Microsoft App Certification 1 67](#_Toc480137947)

[Figure 27 - Microsoft App Certification 2 67](#_Toc480137948)

[Figure 28 - Microsoft App Certification 3 68](#_Toc480137949)

[Figure 29 - Microsoft App Certification 4 68](#_Toc480137950)

ACKNOWLEDGEMENTS

We are very happy to say we completed our project “Brainworks” within the given time by our lecturer Dr. John Healy. We would like to thank John for his seminars and advise on the thesis write up.

Thank you to Dr Martin Kenirons and Damien Costello, our Mobile Applications Development lecturers, for giving us our foundation of knowlege in C#.

We are extremely grateful for our Supervisors Brian McGinley and Martin Hynes for meeting with us weekly to talk about the project and for their helpful guidance and advice.

We would also like to thank all our friends and family, who have been there for us, for moral and financial support which has pushed us through to completing our studies this far. We couldn’t have come this far without you.

Last but not least, to all the students who have either left the course and still remain in the course, you all have helped us get this far in some shape or form. From helping us with our course work, to motivating us to keep pushing forward. Thank you.

INTRODUCTION

Over the last number of years, technological advancements have been so great that it allows just about anyone to have some sort of access to it. Whether it’s the wide range of Smartphones and Computers, to common household staples such as Washing Machines and Televisions, you will still see it in some shape or form wherever you go. This has allowed for highly technical devices to be accessible to everyone, which was not at all the case back in the 1970’s and 1980’s when computers and phones were not common at all among the general consumers, with computers costing far more than they would today. For example, in 1972, the most basic model of an HP 3000 sold for $95,000, the equivalent of slightly over half a million in today’s dollars. *(Evan Comen, 2016)* In the year 2017, the average cost of a laptop was between €300 and €700 (Piltch, 2016) and have a whole host of brilliant features, from microphones and cameras for applications like Skype, to RGB Keyboards to allow the user to be able to type in all lighting.

One particular area of computing that has been touted as having many benefits has been applications that deal with training the Brain. Whether it is about teaching young children and working their apparent “sponge brains” *(Nadia, 1993)* to work at a more proficient capacity to others who may have suffered brain injuries due to health problems or perhaps from a damaging accident. Although there have been a small number of claims and papers carried out saying that there is no groundbreaking evidence that brain training applications actually aid in increasing the ability of the brain, the majority of scientific bodies and those who are given these training sessions have said that there have been many benefits to using such applications *(Daniel J. Simons, 2016).*

With applications out there like Posit Science’s Brain Fitness *(Brain HQ)* and Lumosity *(Lumosity)* requiring a premium to access all of the content, and the popular Nintendo DS game “Dr. Kawashima's Brain Training” requiring the actual cartridge and a console, my team members and I thought we’d develop an app with training the brain in mind for a device that most people already own or have the ability to gain free access to; a PC. We then decided on developing the app as a UWP(Universal Windows Application) so that the application could be run on a wide range of Microsoft-related devices, from Windows Tablets, Phones, Windows 10 Computers and Laptops and even on the Xbox One. We also decided on UWP as Windows was the most popular Desktop and Laptop Operating System having over fifty-percent of the total numbers of Desktop users using their Operating System *(Stack Overflow Developer Survey 2016)*. This means that the common household is more than likely to have at least one device that is able to run our application.

Before we started doing any sort of programming for the UWP, we made sure to have several meetings centered on writing and talking out ideas on what could be done in the timeframe we were given. We decided the best place to do this was to book out one of the study rooms in the College library that had a television to hook up to our laptop via HDMI.

This allowed all team members to be able to see what was being researched and to be able to debate without having to crowd around a single computer. When it came to researching the ideas for the app, the ideas were centered on puzzle-type games as at the time we had been playing games like earlier mentioned Lumosity and also QuizUp *(Plain\_Vanilla)*. These kinds of games are easy to play allowing even young children to work on training their minds. We wanted to capture this idea in our application, by making the user learn by playing the game with no complicated instructions and getting a feel for them that way.

By having an idea of the goals of our application, we started planning out how it would be programmed. This included what IDE(Integrated Development Environment) we would use and the programming language we would use. Since we had previously worked with Microsoft’s Visual Studio in an extensive capacity, we thought it would be smart to work on the project using that.

When researching the languages that UWP supported, we learned that it supported C# or Visual Basic with XAML, JavaScript with HTML, or C++ with DirectX and/or Extensible Application Markup Language (What's a Universal Windows Platform (UWP) app?, 2017)

Figure - Universal Windows Application

Since all of us on the team had already worked previously on C-Sharp(stylized as C#) projects, we decided that working with C# would be the most proficient option, allowing us to fine-tune our more obvious skills first before delving into other languages. We decided, although there is now a Microsoft Visual Studio 2017, we would use the more stable 2015 version.

We as a team of four members have many vastly different ideas and thought processes, and this leads to ideas colliding and ideas shifting and changing throughout the entire project. This can lead to many ideas and plans being pushed to later dates which can lead to them never being fully realised. This to us is a major indicator of failure. The other aspects of the project that we all agree on and have been implemented properly without anyone having any hesitation is what we can easily consider a success for the entire experience.

All of the code, the README and any other file created for this project is hosted on a website called Github. GitHub is a Git repository hosting service, but it adds many of its own features. While Git is a command line tool, GitHub provides a Web-based graphical interface. It also provides access control and several collaboration features, such as a wikis and basic task management tools for every project. *(Finley, 2012)*

In our GitHub repository, the first folder you see is called “4thYearAppliedProject”. This contains all the code necessary in order to execute the Brainworks Application. Following on from that we see a directory called “Website” which contains the Go file(main.go - contains all backend code necessary for website) and all the html (signup.html and login.html – contains login/signup frontend code, Internal.html – frontend code which is only accessible when signed in, Index.html, which has the frontend code for a user who visits the page for the first time or has signed out and finally the Scores.html file isn’t necessary as we eventually decided upon html templating embedded within the Golang itself) files that can be viewed from our hosted website. The “packages” directory contains all the extra files we downloaded through Visual Studio to create functionality, such as accessing SQLite database using Azure Mobile Quickstart. The .gitignore file can be added when creating the repository for the first time. It is used to remove excess/dummy files you might not need which are generated without the developer’s knowledge. The next two; 4thYearAppliedProject.sln and 4thYearAppliedProject.suo are automatically generated by Visual Studio when you create a new project. README.md just contains a brief summary of the project. TeamMembersJobsSpecification.xlsx is an excel spread sheet containing how the project work would be divided between the members of the team. Finally, prezi.pdf contains the presentation we gave to our class back in October which has early ideas we had on the project. The only thing you may see added in the future is a short video presentation going through all aspects in the project.

Our Github account can be found at:

[*https://github.com/ClaireFinn74/Applied-Project-4th-Year*](https://github.com/ClaireFinn74/Applied-Project-4th-Year)

Running the application:

Once you are on our GitHub link, you will see a green button labelled “Clone or download”. Click on it and then click on the “Download Zip” link to download the application. After you download the entire application (approximately 200MB in total), unzip it using software such as 7-Zip or WinRAR to unpackage the files to a directory of your choice. Once that’s finished, navigate to the directory and inside the folder should contain a file called “4thYearAppliedProject.sln”. Double-click on this and it should open up Visual Studio 2015 with the project inside. Once the app is set up, click the button near the top of the screen with the green arrow labelled “Local Machine”. Wait a few seconds and the app should then be fully playable. Another option is to download the application through the Windows App Store. At the time of writing this we are in the process of submitting the application and have no current URL to link to it, this will be updated once the app has been successfully submitted.

METHODOLOGY

The methodology, or system of methods used in order to carry out and eventually complete the project, is a veryimportant consideration when taking on a project. Do we use a slow, iterative approach with clearly defined objectives at the start like the Waterfall Method? Rigid, unchanging and clear allowing a safely planned project throughout. Or, do we use a very fast-paced methodology like Rapid Application Development (RAD) that uses minimal planning and documentation in favour of rapid prototyping. “A prototype is a working model that is functionally equivalent to a component of the product” *(SDLC-RAD Model).*

Or simply, an adaptive approach where documentation and planning are incorporated along with a more flexible attitude to the planning and implementation?

It was clear from the starting point of this project the methodology that our team members would use for the duration. The team realized, when the objectives were set, that the project required a lot of chopping and changing, reworking and getting elements to fit together with limited knowledge on the how-to of the integration as we chose to do a UWP Application that also had a website attached to it having never before connected the two. What methodology would help us to allow change throughout but also had some structure to it? Certainly not the Waterfall method, with its rigid time-scaling and documentation needing to be set out from the start and remain unchanged.

The RAD approach sounded too dangerous as planning needed to be a big part of the project and so did the documentation.

Agile, from the off, was the methodology decided upon for the project; Adaptive (allowing change and integration), iterative, incremental, documented throughout all stages of development (not just at the start), communicative (team members being able to talk things out throughout and pitch ideas at all stages of development) and also forgiving of change.

So now that we had our methodology, Agile, we just had to implement our project in the classic Agile way. Working software, within Agile, is the primary measure of success meaning that the focus is more on getting the software working correctly than ensuring it is done on time or followed the original plan set in motion. It can be much more helpful to a developer to be constantly receiving feedback from supervisors about their project and thinking of ways to improve it rather than stressing over documentation and deadlines.

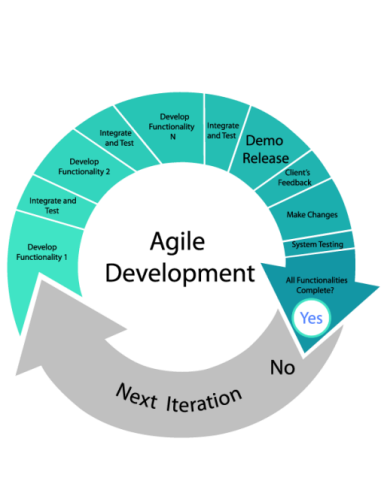


Figure - Agile Development

Proceeding in that vein, we decided with our two Supervisors, Brian McGinley and Martin Hynes, that we would meet up as regularly as once a week, eventually settling on 12pm on a Tuesday as our usual time to receive helpful feedback on the development of the project. Brian and Martin would start off by asking our team what we had achieved the previous week on the development of the project, what we wanted to achieve on the project this week and how we would set about doing it. They would then listen to any problems or questions we had regarding the work we would set out to do that week or any outstanding issues with the previous week’s work. These meetings were a helpful guide in the right direction for our team, motivating us to do well and keeping us on track.

Agile has a framework/technique used within it called Scrum, where the team plans how to develop the product in ‘sprints’ taking a small piece of work from a prioritised list called a ‘backlog’ each time to be completed. A sprint is usually 2-4 weeks and the team must complete the work within that time-limit. The team get together daily to discuss what work has to be completed, and what work needs attention in what is called a ‘stand-up meeting’. The team must have a ‘Scrum-Master’ who keeps the team motivated to do the work. The work should be fully completed at the end of each sprint. The team then moves onto another sprint beginning the ‘Scrum’ Process again.

This technique is one the team incorporated, loosely, into our idea of Agile for our project.

Before we started doing any form of programming for the assignment, we made sure to make plans with our assigned supervisors given to us by the module lecturer, Dr. John Healy. Our Scrum-Master being our supervisors, motivating us and keeping us on track, we had daily meetings that could be called ‘Stand-up Meetings’ with the team to discuss the project and worked in sprints of about 2-3 weeks, working on different parts of the project each time a new sprint began. Although we didn’t have an official ‘backlog’, we still had a list of things we needed to complete and we took these pieces of work and developed the project in what can, however loosely, be called ‘sprints’. We didn’t completely use this ‘Scrum’ technique by the book, but in the true Agile fashion we used it and adapted it to our ever-evolving needs.

With Scrum, you heavily rely on communication with your team members to get the job done. Over the course of the assignment, we had to continuously plan meetings to see how each member was getting on implementing previously discussed parts of the project. If anyone was falling behind, the whole team decided we would have a look at wasn’t working for that team member so they wouldn’t fall behind further. If a problem persisted, then we would remove the functionality as it was, and try to find a way that team member could recreate that same functionality using another means so to avoid completely scrapping good work.

This also means we were saving that team-member’s feelings towards the work they had done, as a developer can feel an enormous amount of pride in their part of the work and can almost feel rejected if their work is scrapped *(What is Scrum?*).

Communication involved talking but it mostly involves listening in a non-judgemental way, this was how we as a team would deal with problems in the development of the project, skilfully avoiding confrontation through communication.

The first time we met up with our supervisors, we communicated the ideas that we had for the project and a basic layout of what we planned to have done for the project. After hearing what we had planned, the supervisors interjected with their own ideas on certain elements we should implement while also avoiding confrontation with constructive criticism. For example, they mentioned having more emphasis on using an n-tier architecture to keep the Client side and Server side completely separate from one another (*N-Tier Data Applications Overview*). This backward-forward communication between ourselves in a team of students with no real-world experience with programming and our two supervisors with their industry advice allowed us to keep current and to communicate effectively.

The technique involved also requires software and tools to plan and implement the project.

To keep track of and manage all the changes being made to the project, we made use of an online service called Github. (*GitHub*) This allowed us make commits any time we made a substantial addition to the code and securely save it online.

To make this process even more seamless, we used Git Bash (*Git*), a command-line tool that allows you to upload large files to Github from your device with a few small commands. We then are able to merge the code of the other team member’s accounts to our own and work from that up-to-date version. Although Github is great for having an online developer’s portfolio, it also has some limitations in the fact that there are merge conflicts over minute details leading to frustration for the team wanting to get work done fast. Due to this limitation, the team had to resort to Microsoft Onedrive sharing amongst team members and uploading the project to the Master Branch. It is unfortunate that you cannot see exactly who did what on the project this way but the team can still access this branch and freely download the newer version.

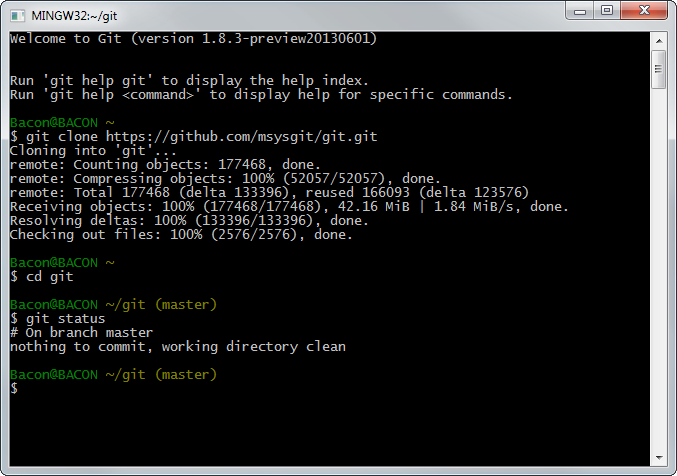


Figure - Git Bash

Github also allows you to give a name and description to each commit, so we can easily find when certain features were added and by whom exactly. This made it easy to see if the designated work given to each member was being committed and when. You also have the ability to write in your repository’s wiki. Here you are able to write details about the current status of the project, mention any problems you may have had that affected the project and mention in more detail goals and milestones you want set out for the project. As we were using an Agile approach, we felt as if the wiki side of Github didn’t suit, requiring too much documentation when we wanted a more communicative approach seeing as we would see each other in person every day. The Wiki side of Github may be more suited to global teams.

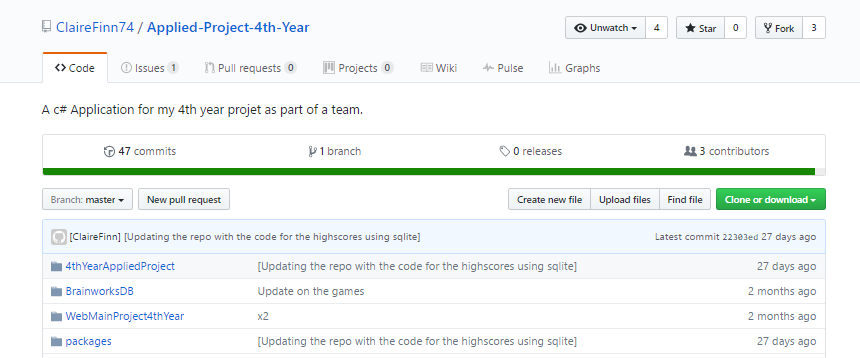


Figure - GitHub

For writing the rough draft of the thesis the team used Google Docs as the document could be modified by all team members and saved automatically.

This made it easier for the team to work on the thesis together and change things whenever we saw fit. You can also see who exactly is editing it in real time.

As for the selection criteria for the languages, algorithms, platforms and technologies used, we all discussed our strong points and what languages or platforms we wished to learn more about. For a start, the four of us felt as if our strong point in languages lay in c# so we used c# as our backend language on our UWP Application. The UWP application itself was decided upon as the four of us felt that Microsoft had the highest global reach in terms of devices as at least one person will have a Microsoft device in a household whether it be a tablet, desktop, laptop, Xbox or phone. The website used MySql as its database and we used Mysql Workbench to manage the data within that database.

We decided amongst ourselves MySql would be our best bet as the website can easily integrate the database. The website also used Golang *(The Go Programming Language*) as its backend language along with HTML, JavaScript, and Bootstrap templates. We decided upon Golang due to its ability to add professional, sleek bootstrapping templates such as Jumbotron, the one we used. Golang is also relatively new as a language and was created by Google, so we felt it would be great to have the experience in this language going forward in our careers. Keeping with the Microsoft theme, we used (*Microsoft Azure*) to host our MySql database as we had a student Azure Pass for free credit, cost obviously being a main concern in a project, was now not an issue with Azure.

As with every project there were some caveats. We strongly wanted to use a restful API called *(SlashDB*) and invested a large amount of our time integrating it into our project. SlashDB connected to our MySql Database seamlessly on Azure but fell into some problems regarding our UWP Application.

We could issue a get request to our database on Azure through UWP and receive the data already within the database but we could not get a post request to integrate correctly with the API through UWP to post information to the database as we needed to post up high-scores from our games. In the end, we decided to scrap any added complexity and use something that integrates seamlessly with UWP through Azure for that particular piece of work, SQLite. We could not afford to lose any more time to the REST API there but we felt we could use it instead to get a list of users that were logged into our app for administrative purposes, again, not wanting to scrap good work.

The project also required testing throughout. To test our API, SlashDB, we needed to check the validity of our HTTP requests from the UWP app up to SlashDB. As SlashDB was using a Ubuntu Server on Azure we need to make sure the server and SlashDB were communicating correctly. For this, we used (*Hurl.it*). We could ask Hurl.it to do a Get Request, put in the link address to our API, which is called SlashDB, and send the request. Then we would get back information from our server or if not, we needed to delve in further to check any errors. Thankfully, we got all of the information back from the server.

You could also do Post requests to send information to the server on Hurl.it. Another form of testing done was with (MySql Workbench). As we were using a MySql database on Azure to store users for login to our Website, we needed to check if the users were being inserted into the database correctly every time a new user signed up and if the MySql database was being validated against for existing user login.

MySql Workbench allows you to see exactly what users are in the database and you could also manually insert users through SQL commands.

TECHNOLOGY REVIEW

The technologies used within this project:

Platforms:

\* Visual Studio 2015 – Universal Windows Application Platform

\* Microsoft Azure Cloud Computing Platform

IDEs:

\* Visual Studio Code

\* MySQL Workbench

\* Visual Studio 2015

\* Microsoft Blend for Visual Studio (User Interface)

\* CMDR Console Emulator for Windows

Databases:

\* Sql Database on Azure

\* MySQL Database on Azure

Servers:

\* Ubuntu Server on Azure

\* SQL Server on Azure

Testing:

\* MySQL Workbench

\* Hurl.It

\* Selenium

\* Microsoft App Cert Kit

Other Technologies:

\* Ubuntu Virtual Machine on Azure

\* SlashDB API

\* Swagger UI API

\* App Service on Azure (Facebook Configured)

\* GitHub (Version Control and internet hosting)

Languages:

\* C#

\* Extensible Application Markup Language(XAML)

\* Golang

\* Html

\* JavaScript

\* JSON

\* SQL

\* Bootstrap/CSS

Universal Windows Application:

As set out in the objectives of the project in the Introduction, we, as a team, aim to create a Brain-Training application consisting of four sections: Attention, Memory, Problem-Solving and Speed. To do this, we created a new Universal Windows Application in Visual Studio 2015 and began designing the front end, each of the four of us on a different section.

We used XAML to design the User Interface, complimenting this with the use of Microsoft Blend for Visual Studio to create Storyboards and Animations to increase the look and feel of the games.

Extensible Application Markup Language, or XAML, is an XML-based markup language developed by Microsoft. XAML is the language behind the User Interface/visual presentation of an application that you develop in Microsoft Expression Blend or within Visual Studio itself. (*Microsoft Blend for Visual Studio 2015*)

The back-end code to the games was built in c#(pronounced as “see-sharp”), an Object-Oriented language developed by Microsoft in 2000, and of the C family. C# is approved as a standard by ECMA (*Standard ECMA-334*) and ISO (*ISO/IEC 23270:2006*). C# is the most widely used Common Language Infrastructure language (*CLI*). CLI allows multiple high-level languages to be used on different platforms and can be considered ‘Platform agnostic’ (*What is Platform Agnostic?*). ECMA listed a number of design goals for C#: One of the standards stated that C# is intended to be suitable for writing applications for hosted systems.

C# was used for almost the entirety of the backend programming of the games. The games also need high-scores. One of the main caveats to Universal Applications is the fact that you cannot directly access relational databases using technologies provided by Microsoft. This can be seen as a caveat, but it can also be seen as a good thing as the lack of third-party software can make it easier to download the app as it would not have a reliance on other services.

Some third-party software would be a solution to this, our main reason for selecting SlashDB, which automatically generates a REST API from relational databases like MYSQL. (*SlashDB*)

You can then read/write data in JSON, XML or CSV. **SlashDB** works in two ways: Data Discovery, which automatically reflects all objects in a database as URL addresses and ‘SQL Pass-thru’ which configures any SQL Statement and ‘maps’ it to a URL. In ‘Data Discovery’ Mode, all records automatically become HTTP resources which can be read and updated. As we had n-tier architecture in mind for the project, we felt SlashDB would be the correct way to go about achieving this by hooking up a MYSQL database, created on Azure, to SlashDB and then writing some JSON Code within our UWP Application to access the data on MySQL.

SlashDB is a great resource for people who need an API fast. The only disadvantage is the fact that there is relatively no information on SlashDB other than on the main SlashDB website.

We received lots of advice from a friend in industry on how to use SlashDB correctly. We also researched extensively into this area before we began as we had never attempted anything of this scope before( (*SO\_Questions*) (*Json Serialization and DeSerialization*) *( Retrieve data from URL into JSON*) *(Return JSON file with ASP.NET Web API*).

We could issue “GET” requests just fine to SlashDB and retrieve the JSON data back from our MySQL database but found it difficult to actually “POST” information to the database. The code used a JSON URL that was generated for us from SlashDB and this allowed us to pull through the JSON data which contained the list of users but when trying to use a “POST” method to send our new high scores to the database we encountered several errors saying the data we were passing to the database did not match the data that was already present there.

This is a main concern, how are going to pass our high scores to SlashDB? We eventually achieved “GET” requests to the database through JSON in our UWP application but we spent a lot of time researching this part of the project and asking questions to stack overflow in regards to how we could possibly work this part of the project to our own use *(Error on getting one part of JSON* ) *(Trying to return users from a database in a textbox*) (*Displaying a part of the JSON returned from a DB in UWP)* but the only information in that database was a list of users already registered with our website.

We first thought about adding a column to this for high scores but in the end because of the difficulty we encountered with getting post requests to work with our MySQL database we decided we would leave this feature as a “GET” request for a list of users. This did not solve our high scores problem, but it did serve well both for administrative purposes to see who was registered with the website, and to allow users to check if they are registered with the website themselves (The website will be discussed further on).

When designing the code for getting our list of users we spent a lot of time looking at how the JSON could be parsed into our app and then we would display it in the app by a single value – username. We discovered that we did not need to go this far but instead we could use the code that we had already designed for reading the JSON data into our app that contained all the info in the database – username, password, etc. This worked off using the URL created by slashDB and parsing the JSON info in from the web api to our app. It was slashDB that was vital here as it allowed us to create an SQL statement which limited the data down to just username and left everything else out, which is what we needed to provide security to our registered users with our website otherwise their passwords would be vulnerable. Using said SQL statement slashDB mapped the new JSON data (only username) to a URL and that is the URL we used in our app to allow users to see if they are in fact registered with our website.

However, this was still only a “GET” request to our mySQL database and after some extensive research that cost the team time on the project, we decided to do some deep consideration on the advantages of continuing to search for reasons why our “POST” requests weren’t working. We felt the lack of documentation meant that we would not find a solution on time. We then thought what database could be used with our application that seamlessly integrated on a time crunch?

Microsoft has a lightweight database called SQLite. It is said to be ‘Zero-Configuration, server-less, lightweight’ (*SQLite Overview*).

This didn’t fit in to the n-tier architecture we had in mind, but on further thought, we concluded that we could at least use an ‘App Service’ on Azure to host our SQL Database on the cloud. This would then solve our above problem. In order to utilize this App Service, we decided we could use Microsoft Azure’s ‘Mobile Apps Quick-Start’ option, which generates a backend for your Mobile App, creates a table and gives some starter code to incorporate into your mobile application code.

Upon incorporating the code from the Quick-Start option on Azure, the app now had a fully functioning backend with scores updating and being stored on Azure all from a simple URL. Now, for a while we didn’t understand how the backend was being generated but we needed to find this out so that our website could access the scores. We wanted to be able to display these scores on our website therefore rounding up the architecture we desired (n-tier). We eventually figured out how things were being done on Azure, to be discussed in the Website section below.

Another feature of our application is Facebook Authentication. We achieved this functionality by using the same App Service we used for our high-scores. To configure Facebook Authentication you need a Facebook developer account (Facebook Developer).

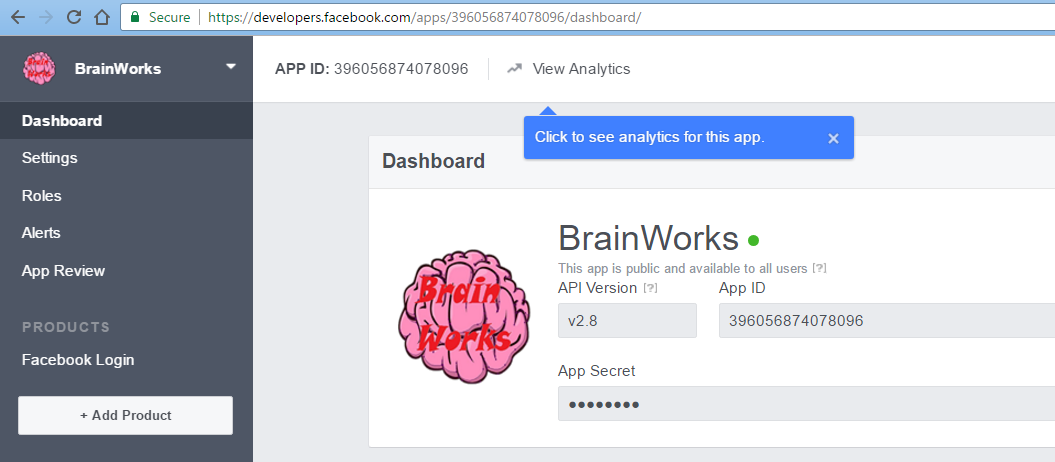


Figure - Facebook Developer

As you can see from the image, a Display Name for the application, an App ID, and an App Secret is created. You also enter in the URL that your App Service on Azure provides with the extension “/.auth/login/facebook/callback”.

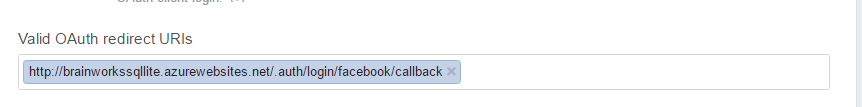
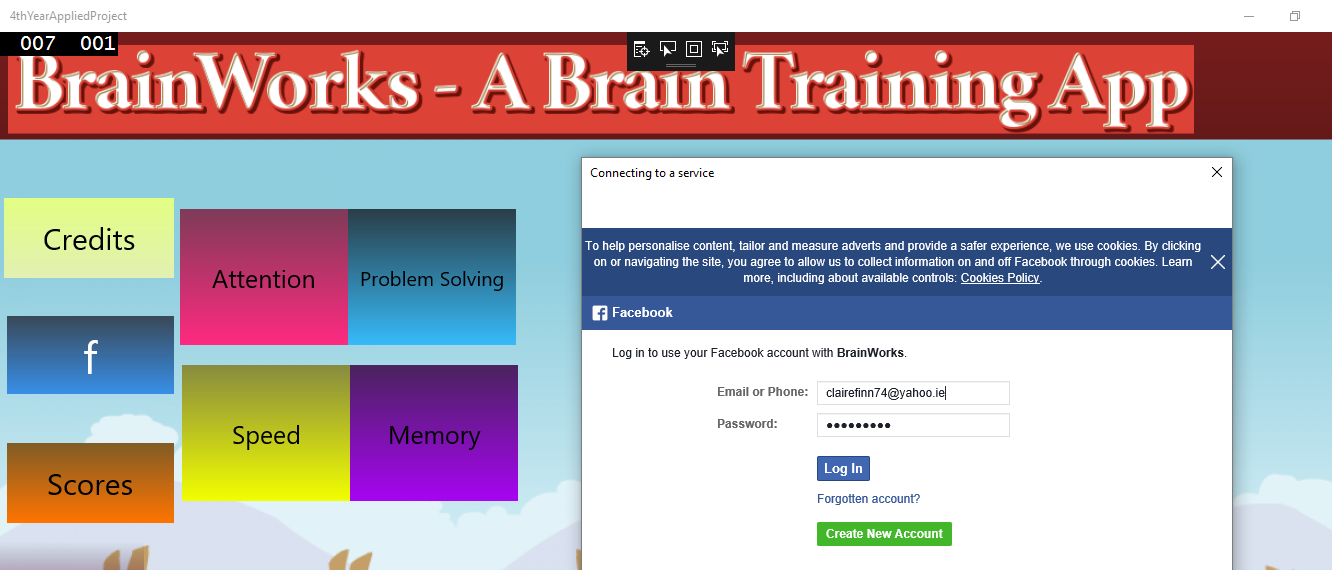


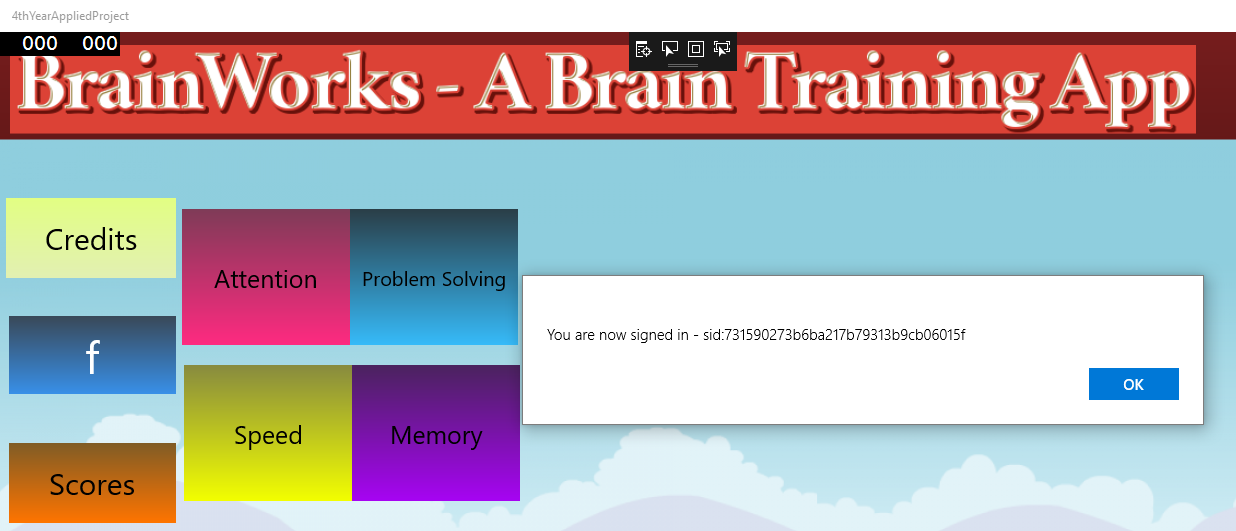
Figure - Valid OAuth redirect URI

OAuth is an ‘open standard’, and is used as a way for users to authorize applications with a third party without giving away their passwords. OAuth is designed to work specifically with HTTP allowing ‘Access Tokens’ to be given to a third party client by an ‘authorization server’. The third party (Facebook in this case) then uses the token to access the resources which are seen as ‘protected’ on the resource server. There is no exchange of passwords this way.

This allows you to connect your UWP Application to Facebook through the App Service on Azure. Once you enter in the App Service URL on the Developer Account you can go into the ‘Authentication/Authorization’ section on your App Service on Azure and select ‘App Service Authentication On’. You can then choose to authenticate with a number of services: Facebook, Twitter, Microsoft Outlook, Azure Active Directory or Google.

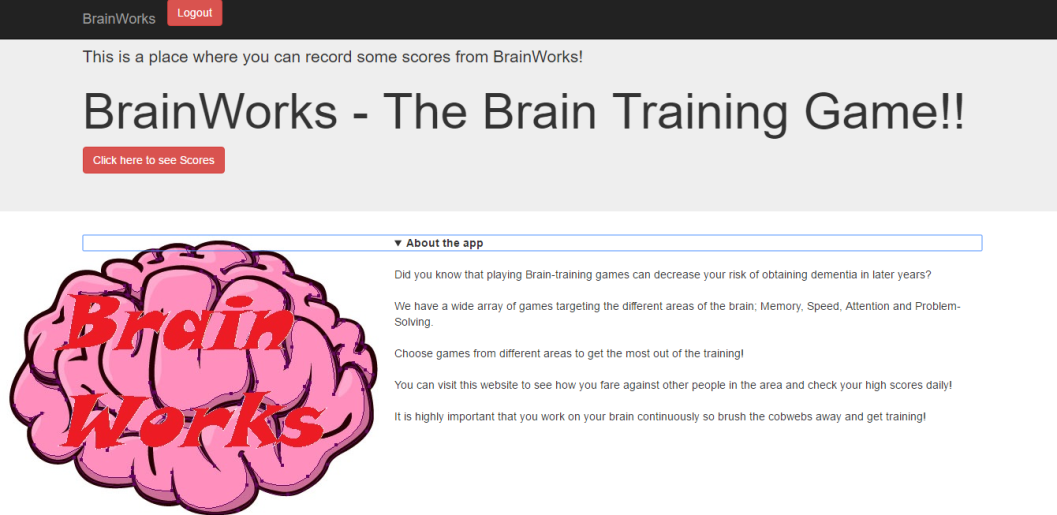
As we had chosen Facebook, we selected that option. Once you select your service there are different rules for each. For Facebook, you enter in the App Id and App Secret you were provided with on the Facebook Developer site. You can then use some code on your Mobile App backend to access this, using the App Service URL. You should then be met with the service provider login interface:

Figure - Facebook Authentication

Figure - Login Successful

You can choose to authenticate solely with the provider you chose on Azure in order to be able to use your UWP application but the team decided to leave the login as an option for the user.

Website:

Figure - Internal Page

The website side of the project was developed as a way to store our high-scores, again, with an n-tier client-server architecture in mind. We decided upon Golang as the backend language for the website. We also wanted to use HTML, CSS and JavaScript to design our website. CSS was the 4th most popular programming language and JavaScript was the most popular programming language used for designing websites in 2015 (*Top Programming Languages Used in Web Development*). HTML is the standard mark-up language used for websites and when used alongside CSS and JavaScript “it forms a triad of cornerstone technologies for the World Wide Web” (*Flanagan*).

When designing the website, we believed the layout should be visibly appealing for visitors. Our internal page reflects this as it was designed to have a natural flow for a user to follow. While reading, the visitor will notice that the layout goes from an image on the right side (the images are hosted on a site called (*Imgur*) which gives us URLs for each image) with its details on the left to vice versa on the next item and then back again to the original format and then continues down the page in a similar fashion on the website (*How to put a text beside the image (html)?*). We believed this style to be more appealing and eye-catching to look at as a design choice rather than having all the images on the left and all the descriptions on the right. To de-clutter the look we also included drop downs so visitors can remove what they’ve already read by re-clicking on the “about the game” drop down so the site has a minimalist view to it.

In each drop-down is a small blurb about each game, describing it and listing its creator. This whole look was built on top of a bootstrap template called ‘Jumbotron’ as we wanted to customize the look ourselves. The layout is handled in html using URLs on the internet for the images and html functions to display these images and their respective drop-down lists.

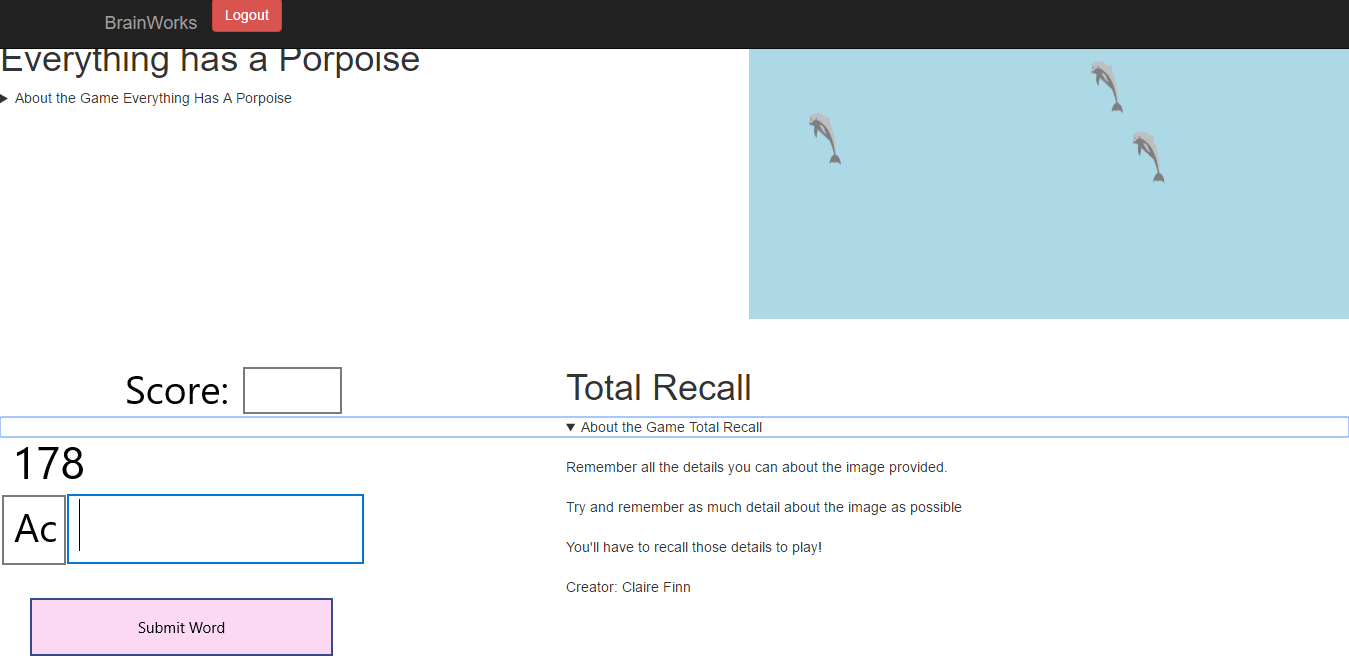


Figure - Example Game Synopsis

The Golang backend handles users logging in and signing up to the website using a MySQL database on Azure. By doing this, we are able to see how many unique visitors actually visit the website, whether it is to view the scores that other users have obtained, even their own scores or they might just want to know more about the games they’re playing through the game synopses on the Internal page of the hosted website.

In order to handle users trying to sign up and log in, we created Login.html and Signup.html pages that respectively contain a POST method called “/signup” and “/login”. These methods allow you to group together certain elements from a web page, for example, in our website we have the username, password and submit elements within both the login and signup POST methods found in both of their html pages.

These POST methods are then controlled by the main.go file which does all the backend handling for the website. The Golang file carries out error handling for the website, such as informing the user that their credentials could not be entered into the database due to them not having their internet turned on after navigating to the page for example. It also allows you to use SQL statements to search a database. We used this to search and insert into the database the information that users input into the textboxes on the website. You can also use import statements in your Golang file. One such example in our Go file is the “bcrypt” import. “Bcrypt is a cross platform file encryption utility. Encrypted files are portable across all supported operating systems and processors. Passphrases must be between 8 and 56 characters and are hashed internally to a 448 bit key. However, all characters supplied are significant. The stronger your passphrase, the more secure your data” (*Bcrypt*). This prevents anyone other than the user being able to see what the user has set for their password, and as mentioned above, greatly improves security for any user who sets up an account of the website.

In order to display the scores in the website we have to use the App Service described in the UWP section above. As mentioned earlier, we had no idea how we would access the scores from Golang. On further inspection, we managed to discover a backend script which allows you to ‘run’ the app service. We were redirected to the app service URL which had a ‘try it out’ link. After clicking this link we were brought to Swagger UI. After some research, we found out that Swagger UI is an ‘OpenAPI Specification’ for consuming, visualizing and describing RESTful web services. From here, we were able to see exactly what HTTP requests we could make to obtain access to our scores.

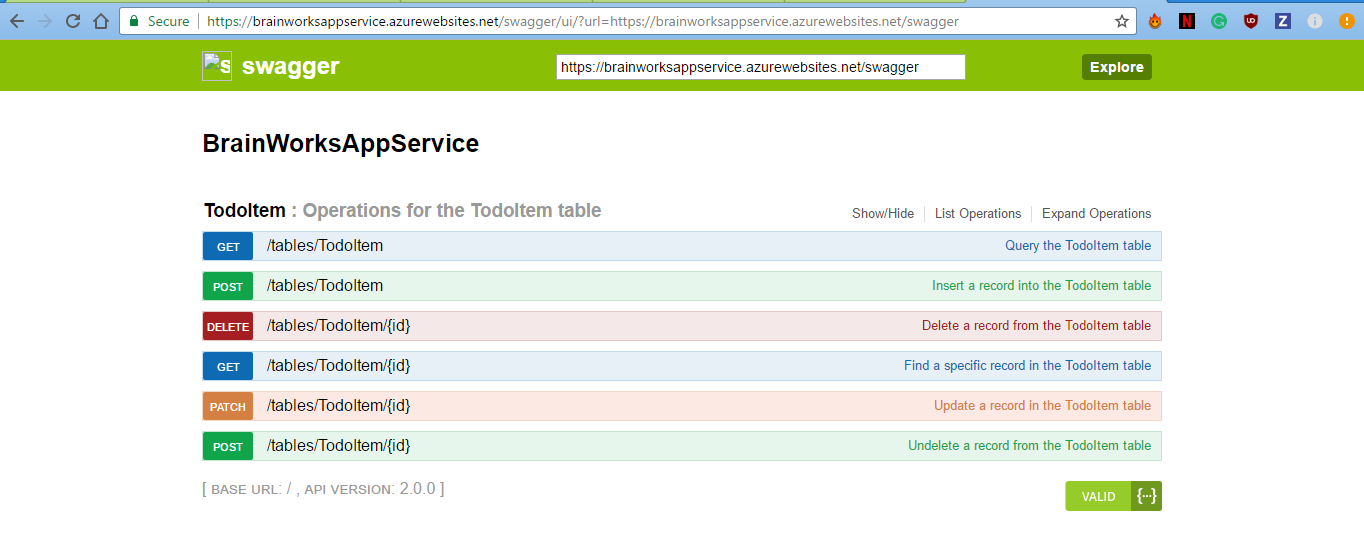


Figure – Swagger UI

We chose the ‘GET’ request and specified $select score as the parameter string to the GET request on Swagger UI and then chose the 'Try it out' button. A URL is then generated for you. It is this URL we can use within Golang to access the scores.

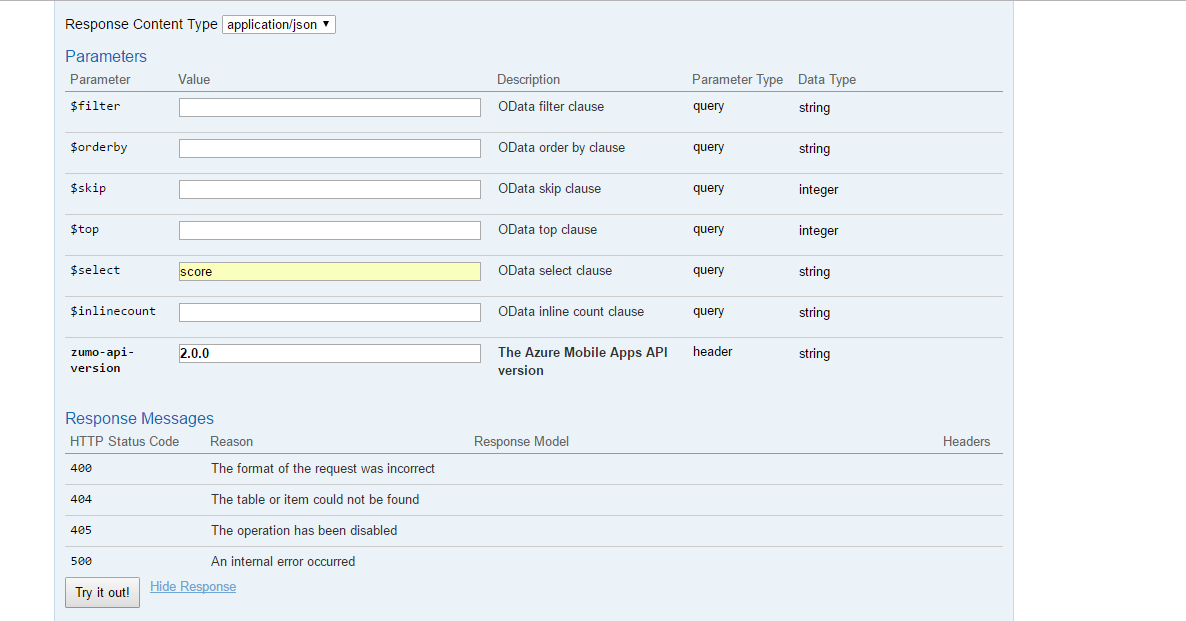


Figure - Search score using Swagger UI

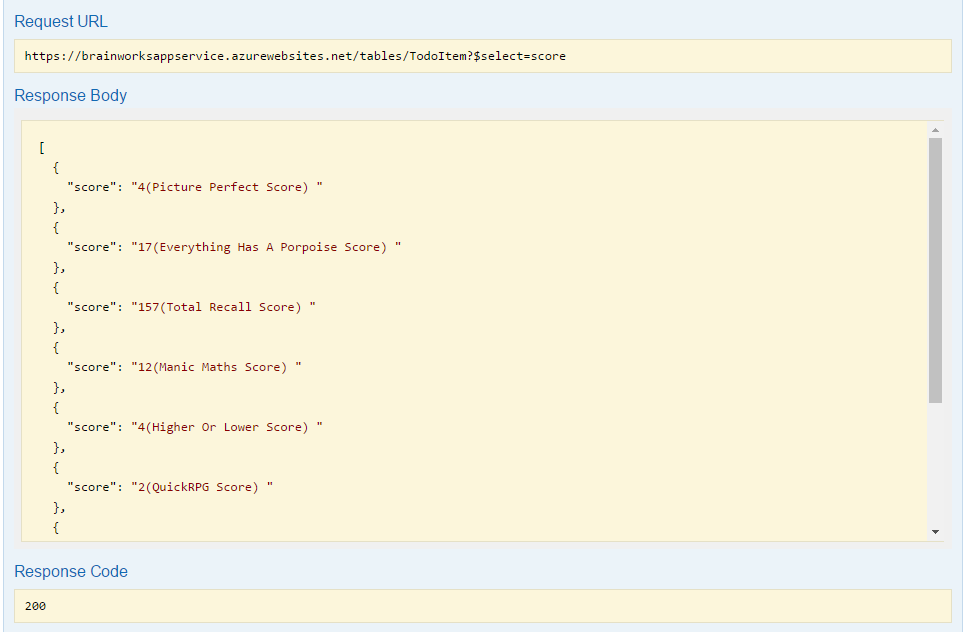


Figure - Displaying info from DB from Swagger UI

In Golang, you can issue a JSON ‘GET’ request from your website to the URL described above.

You can then print this out to the console (we used CMDER, “Cmder is a software package created out of pure frustration over the absence of nice console emulators on Windows.” (*Cmder*). With CMDER, you can also use git-for-windows, which allows you to use GitHub commands all in the one console) for testing purposes but that simple HTTP request in Golang will not be enough to display the scores from our UWP Application using the App Service on Azure. With Golang, you can use ‘HTML Templates’. Instead of the usual way you would allow Golang to communicate with HTML (Through post actions in your HTML file to the Golang file and serving up the HTML files on the Golang side) you can use these HTML templates.

HTML templates allow you to specify the design of your HTML page within the Golang file itself which makes it easier to work with the Golang function you created to access the scores. You can do this within a Golang variable like so:

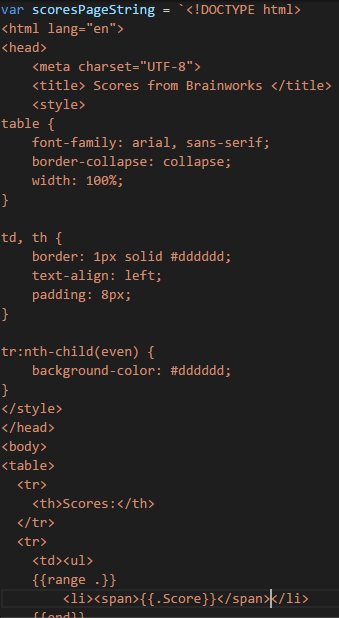


Figure - Sample HTML

As you can see I’ve put the Scores into a table and added a bullet-point structure to make the JSON output look tidy. You must first specify a ‘struct’ which mimics the way your table is formatted on Azure in order to be able to access the fields of your table within the HTML:

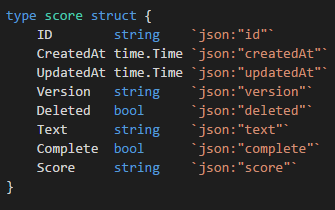


Figure - Struct in Golang

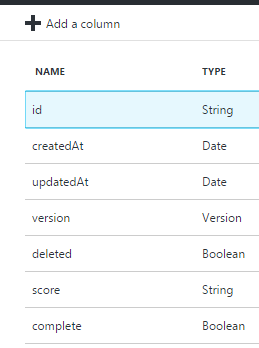
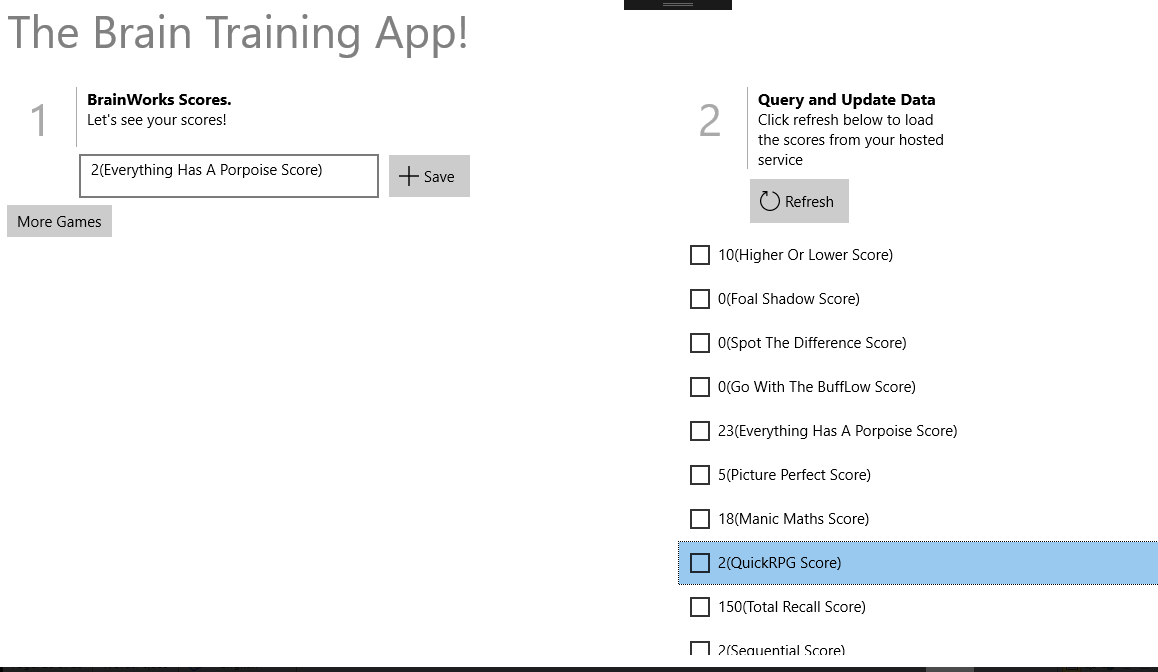


Figure - Structure of Table on Azure

The score within the HTML is then denoted by parenthesis like so: {{.score}} using your struct for reference. As long as you are still pulling your non-static JSON in with a GET request you can convert this for use within the HTML and have a fully dynamic JSON request pulling information from your UWP Application, through the App Service on Azure and into your website! Now every time you update your scores in your UWP Application you can reload your website page and see the scores reflected back instantly.

Figure - BrainWorks Games Scores on the UWP Side

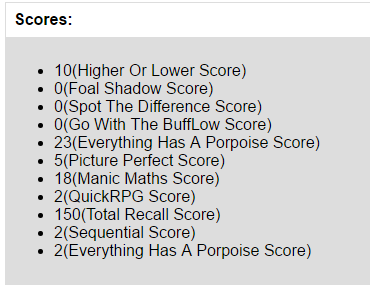
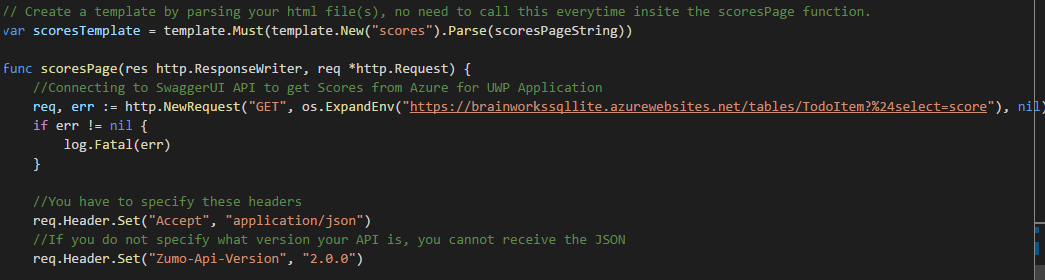


Figure 18 - BrainWorks Games Scores on the Golang Website Side

The only other thing to note is the fact that you must always specify the following headers within your Golang code to access your information using Easy Tables on Azure;

Figure - Golang code to access the scores

If you do not specify the version of your API your website cannot access the URL within the GET request.

SYSTEM DESIGN

What Is System Design? One of the most interesting, and most difficult, of the tasks that we may undertake in our careers as software developers and programmers is the design of an entire system. A system is a set of interacting and interlaced sections, generally too large to be built by a single person, made with a specific function in mind. We work with systems all the time. The operating systems that control our machines are systems, such as the most commonly used three today; Windows, Mac OSX and Linux. The layers of hardware and software that allow the programs on these machines to interact with each other over a network are systems. Even most applications that we use are systems, whether we know it or not (*Waldo, 2006*).

At a much smaller scale, we will look into a more local system and its design with our project. Although the project we worked on did not involve creating a new OS, it still involved several vitally important components that were selectively chosen in order to create and complete the project. Any removal of the major components would result in the application falling down completely. It would be like removing System32 from any Windows computers.

Before we started even thinking about which programming language we would work with or the IDE we would pick, we had a meeting with our supervisors to discuss any ideas we had for our project to get a clearer image for our scope. We were told that if our project was to be completed as solo work, or that one person was to work on it, then that person would need to contribute roughly fifteen hours a week to the project. Since we were working as a four-person team, we were told that our combined work was to amount to roughly sixty hours. Upon initially hearing this information, we all felt that this was a large amount of time to be working on the project, as we had other deadlines and continuous assessments on the horizon to worry about as well, but as the weeks progressed it didn’t feel as stressful as we got used to the workload that was given to us.

From the initial meetings, which mostly discussed what we had been researching and developing the skeleton of the app to build from until we had fully decided on the Software Design we would work from. From then we worked on the app using C# which was developed using Microsoft’s Visual Studio 2015. We then used the program to develop a Universal Windows Application (UWP) so that when the application was finished it could be accessed on a number of devices with minimal differences. Making the application in UWP also allowed us see how the application looked on a number of differently sized devices, from smaller phones to larger tablets and computer screens.

To work with an n-tier or client-server architecture, which is one of the requirements of the project, we decided upon using an SQLite database in order have our application communicate with a database on a server hosted on Azure.

There was a lot of trial and error when it came to the project. Initially stated plans early on had to be changed as functionality we had planned to incorporate had to be scrapped or we had to investigate many different sources until we could find an alternative. One such example was the Mysql.dll that needed to be incorporated into the application through Visual Studio. Whenever we tried using it, we would always receive an error that prevented us from using the .dll. A DLL is a library that contains code and data that can be used in multiple instances at once. For example, in Windows operating systems, the Comdlg32 DLL performs common dialog box related functions. Therefore, each program can use the functionality that is contained in this DLL to implement an ‘open dialog box’. This helps promote code reuse and make the program as a whole run much more efficiently (*What is a DLL?)*.

This is the main reason for us moving over to an SQLite database instead of a more complex MySQL one. Due to time constraints already in place to finish the application, we couldn’t spend a large amount of our time figuring out an efficient workaround in time.

Coupling is the act of joining two or more objects together. In software development, coupling refers to the degree to which software components are dependent upon each other. For instance, in a tightly-coupled architecture, each component and its associated components must be present in order for code to be executed or compiled. In a loosely-coupled architecture, components can remain autonomous and allow middleware software to manage communication between them *(What is Coupling?*). Microsoft’s Azure Cloud Services act as a middleware for our project as it allows for the deployment of databases and servers, as well as virtual machines for hosting, without having to program the entire thing yourself. This makes it much easier and less time-consuming to set up services required for the application without having to create unnecessary complexity (*Middleware Solutions for the Enterprise)*.

Using Azure as our Cloud Platform allows it to also be known as an IaaS or Infrastructure as a Service. This is due to it providing building blocks of computing resources, taking the traditional physical computer hardware, such as servers, storage arrays, and networking, and letting you build virtual infrastructure that mimics these resources, but can be reconfigured within moments.

Azure can also be called a PaaS or Platform as a Service due to it operating at the layer above raw computing hardware.

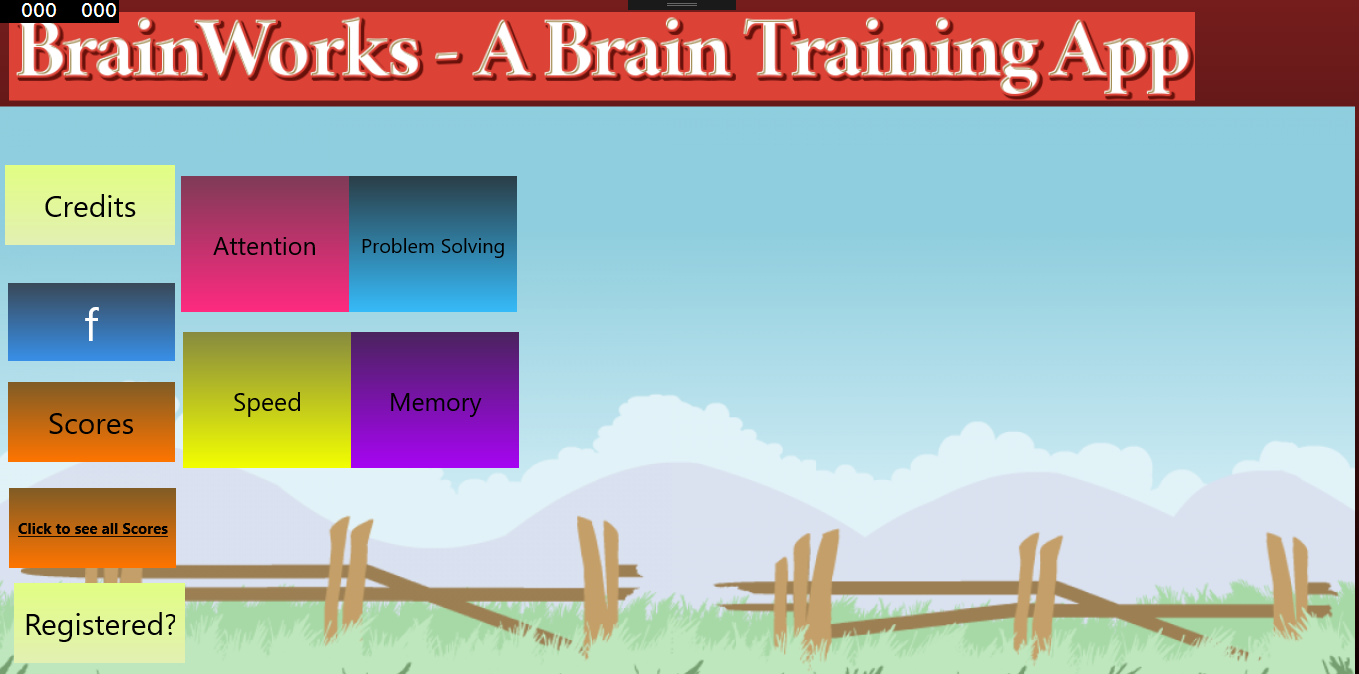
PaaS provides a method for programming languages to interact with services like databases and servers without having to deal with lower level requirements like how much space a database needs, whether the data must be protected or distributing the workload across servers that can be spread throughout the world (*What are basic differences between IaaS, PaaS and SaaS?*).

We designed this project to be very user friendly; the UWP application is laid out in an ergonomic fashion that lets the user easily navigate the program. We believe that this is most beneficial to our users.

The main page has stackpanels that contains eight buttons, in the first column there is a button that will show the user the credits, a button that lets the user login using Facebook authentication, a button that shows the user the scores already recorded and a button that navigates the user to the app’s website.

The second and third columns contain buttons that navigate the user to the game selection for the following categories: Attention, Memory, Problem Solving and Speed.

Figure - Application Main Page



The Attention Games menu page has three buttons to navigate the user to the available games for the Attention category. These games are: “Everything has a Porpoise”, “Total Recall”, “Picture Perfect”. The aim is to increase the users’ attention to detail with challenges that the user must get through while being timed.

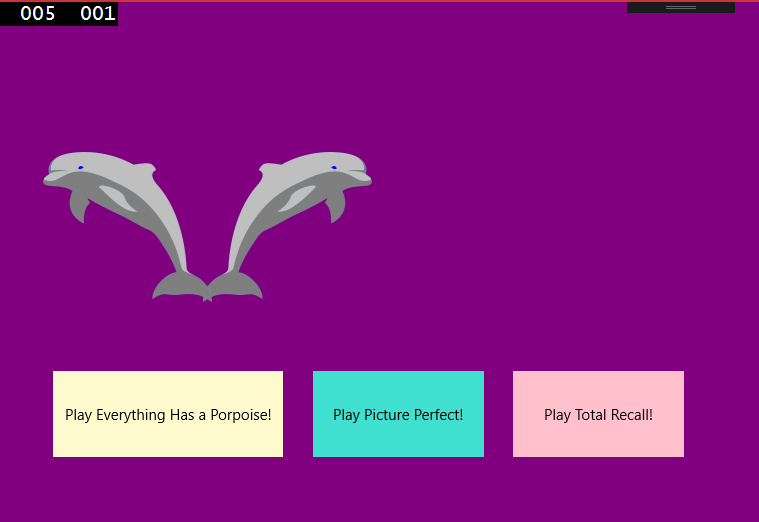
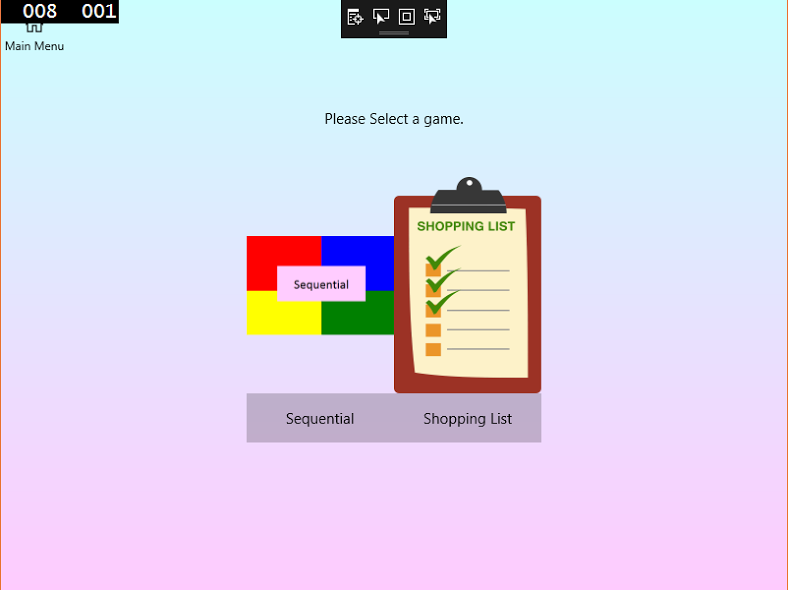


Figure - Attention Page

The Memory Game’s menu page has two buttons to navigate the user to the available games for the Memory category. These games are: “Sequential” and “Shopping List”. They aim to improve the users’ memory gradually increasing the length of items to remember in order.



The Problem Solving Games menu page has three buttons to navigate the user to the available games for the Problem Solving category. These games are: “Higher or Lower”, “Quick RPG”, “Manic Maths”. These aim to boost the users’ problem solving skills by having the user work their way through a problem with a limited amount of time.

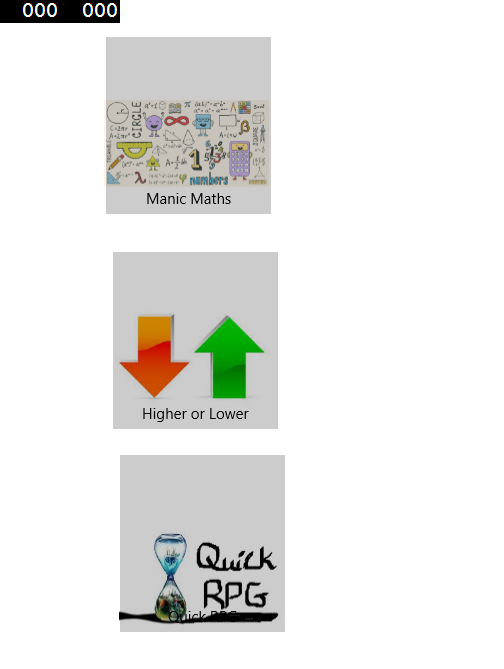


Figure - Problem Solving Page

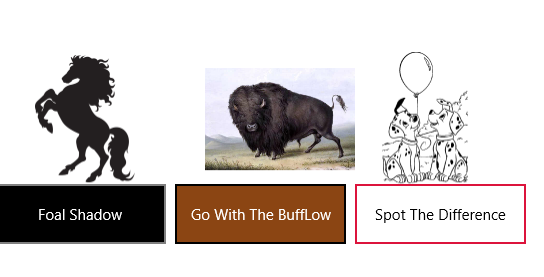


Figure - Speed Game's Page

The Speed Games menu page has three buttons to navigate the user to the available games for the Speed category. These games are: “Foal Shadow”, “Spot the Difference”, and “Go with the BuffLow”.

These games will help the user to build upon their mental speed. These timed challenges aim to push the user to be swift but careful in their thinking.

After each game the user can record their score which is saved to a database in Azure. These scores can be viewed in-app or on the website.

Website UI:

The website’s look is adapted from a bootstrap template called “Jumbotron”, we believe that this template offers us the best aesthetics’ while being thoroughly customisable. The homepage gives the user the option to Sign up or Log In, there are two buttons at the top of the screen that the user can use to choose either option.

On the login page, the user may input their username and password. Should these already exist in the users’ database on Azure, the user is then redirected to the internal page of the website. Otherwise the login page is reloaded and the user is informed that the username and/or password were incorrect.

On the sign-up page the user may input their desired credentials to be added to the database. Once complete, the user is notified and can then navigate to the login page to gain access to the internal site.

We designed the internal page to showcase each game in the application, screenshots, and a short description of the games are housed here. The user may also view the scores saved to the database by pressing the button “Click here to see scores”. The logout button at the top of the page will log the user out and send them out of the internal site and back to the home page.

Cloud Hosting:

This system utilizes Azure, an IaaS (Infrastructure as a Service). Our use of Azure gives us the benefit of building a virtual infrastructure of servers, storage arrays and networks that can be quickly reconfigured from one of our laptops on a centralised website, without the hassle of obtaining and maintaining the physical hardware ourselves.

Website Hosting:

During the process of hosting the website there was a number of unforeseen issues which involved a lot of trial and error in an attempt to resolve the problems. This in itself was quite educational. Golang we quickly learnt was not the easiest language to use when trying to find a site that hosts it. Although Golang has been around since 2011, it is only recently that it has been gaining popularity. Consequently, there are only a select few services that can host a website with Golang.

Originally we had hoped to host our website on a free hosting site *(Free Web Hosting with PHP,MYSQL and cPanel*).

Doing this rather than hosting through Azure, would let us get the site up and running, without the danger of our Azure subscription ending before the project was examined. We soon found out that the average hosting site does not have Golang compatibility, which is crucial to the operation of our website.

We then turned our focus to hosting on Heroku *(Cloud Application Platform*). Many people in our course had used Heroku in their projects and had great success with it. Heroku does have compatibility with Golang, and our website was kept up to date via a local git repository where we could have any new code pushed straight up to Heroku. After a few false starts trying to get our code to push up to Heroku properly, we had successfully deployed our website to Heroku. At this point we reached another obstacle; the website was timing out before it could properly load. When we consulted our supervisors on this issue they advised that we host the website on an Azure virtual machine.

We created our virtual machine and installed all the necessary components to get the server hosting and running our website. This included Golang, Cmder and notepad++. We then set up the server roles and features, this let us set up IIS (*The Official Microsoft IIS Site* ) to host the website from. At this point we tested the website in the localhost of the VM, it was working successfully. When we tested it externally we received an error 405, this meant that the POST method that we were using to sign-up and login was not being accepted.

We spent nearly a week following tutorials, and advice from Stack Overflow *(Asp.NET Web API - 405*), then found the fault in the endpoint of the VM in azure portal. After this we encountered a new error. The website would timeout before a connection could be made to the database that houses the usernames and passwords of anyone wishing to enter the website. According to various sources online, the connection timeout variable cannot be changed on Google Chrome (*How can I change the default website connection timeout in Chrome?*), but is changeable on Mozilla Firefox. We downloaded Firefox on the Virtual Machine and tried running the website. It would timeout before displaying the index page, so we changed the connection timeout variable and still no change.

At this point our azure pass was soon to expire so we recreated the VM in a paid Azure account. We set up everything the same as the first VM. When we attempted to test the website in localhost it would display normally until we tried to sign-up or login. At this point the infamous 405 error returned. This time the problem was not an issue of a port on the Azure portal. Externally the website was simply not displaying at all due to a connection timeout error. We tested the website files to ensure there was not an error in them by running them on our laptop’s localhost. The whole website worked without any error or timeout, all functionality was present and accounted for. This left us with more questions than answers. We consulted with a colleague in industry who has more insight working with Azure. Their insight was highly valued to us and much appreciated, yet the errors persisted.

After some advice from our supervisors, we decided to try changing the port number on our website to port 80 from 8000 and then tested the website by disabling the firewall on our virtual machine. We changed this due to the fact that the virtual machine would look for port 80 by default as most websites on the internet use port 80 for hosting their websites.

After this change the hosted website was fully operational and all its functions were working - login, sign-up, logout, scores, etc.

We then turned the firewall back on again and added both inbound and outbound rules for port 80 and this now allows our website to be truly hosted.

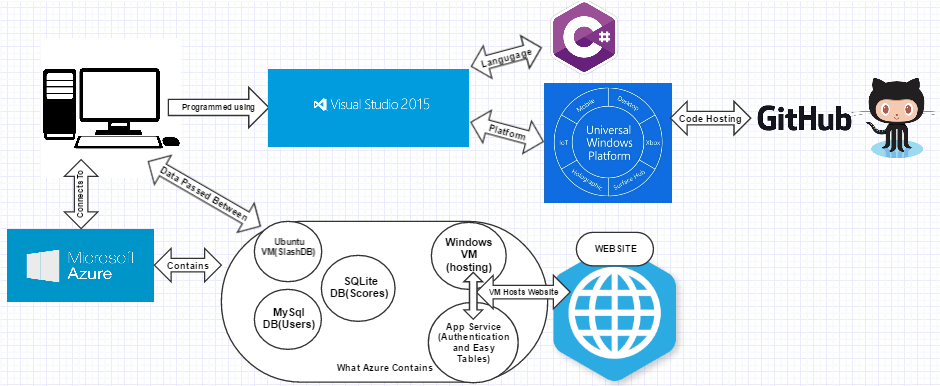


Figure - Project Architecture

SYSTEM EVALUTATION

For evaluating the software we created, it was decided to test our hosted website using selenium test cases and test all of its features through Mozilla Firefox – Drop Down Lists, Login buttons, Logout buttons, Sign-up buttons while also making requirements for the site. We then made a test plan with our results of what worked the way it was expected to and what didn’t. Selenium (*Selenium Tutorial: Project)* itself, is a software-testing framework developed primarily for testing web applications. It can be used as an extension to Mozilla Firefox for easy use and allows for the test cases to be written in a wide array of languages – C#, Java, PHP, Python, Scala, Ruby, Perl and Groovy.

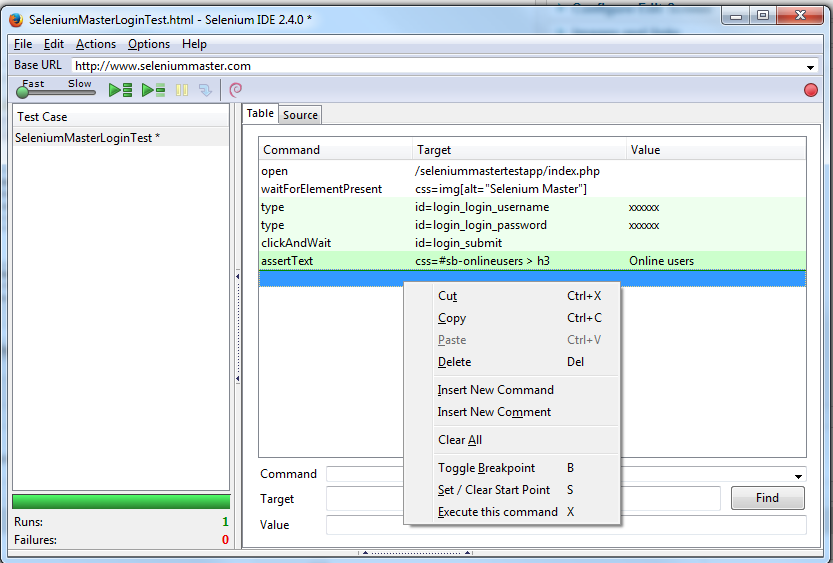


Figure - Selenium Test Case

Selenium is also able to test most modern web browsers. We also decided to list all the requirements of our website that will fall under inspection in our Selenium test cases and thus make a Test Plan to showcase which requirements the website passes and which ones (if any) it fails. We decided for this section to include our requirements here and keep the Test Plan to a separate document.

Brainworks Website:

This is the business requirements for the Brainworks Website

**Requirements 1:** On initial page the game name is located on the left side of the nav bar.

**Requirements 2:** On initial page login button displayed in top nav-bar and brings user to login page.

**Requirements 3:** On initial page sign up button displayed to the right of the Login button and brings user to sign-up page.

**Requirements 4:** On Login page the game name is located on the left side of the nav bar.

**Requirements 5:** On the Login page the user is prompted for a username which is located under the nav-bar and cannot sign in without.

**Requirements 6:** On the Login page the user is prompted for a password which is located under the nav-bar and cannot sign in without.

**Requirements 7:** On the Login page once the user has entered both a username and password they can then use the submit button to access the internal page of the website (assuming they’re registered with the website skip to Requirement 13 otherwise read the next requirement).

**Requirements 8:** On the Login page, there is a display of the current time.

**Requirements 9:** On Sign-Up page the game name is located on the left side of the nav bar.

**Requirements 10:** On the Sign-Up page, there is a username textbox under the Sign Up located under the nav-bar on the left.

**Requirements 11:** On the Sign-Up page, there is a password textbox under the Sign Up located under the nav-bar on the left.

**Requirements 12:** On the Sign-Up page, once both username and password fields are filled in there is a submit button which registers that new user with MySQL database behind the website.

**Requirements 13:** Once logged in, the internal page will now display a logout button in the top nav-bar on the left.

**Requirements 14:** On the internal page, there is a scores button under the nav bar and under the main title (which displays: Ready to see some scores?) which leads to a page displaying scores from the games.

**Requirements 15:** On the internal page, there is a drop-down summary with the title “About the app” located under the scores button which when clicked on drops down more info.

**Requirements 16:** On the internal page, under the “About the app” section there is another drop-down summary called “about the game” for the game – Everything has a Porpoise.

**Requirements 17:** On the internal page, under the “about the game” section for Everything has a Porpoise there is another drop-down summary under that called “about the game” for the game – Total Recall.

**Requirements 18:** On the internal page, under the “about the game” section for Total Recall there is another drop-down summary under that called “about the game” for the game – Picture Perfect.

**Requirements 19:** On the internal page, under the “about the game” section for Picture Perfect there is another drop-down summary under that called “about the game” for the game – Manic Maths.

**Requirements 20:** On the internal page, under the “about the game” section for Manic Maths there is another drop-down summary under that called “about the game” for the game – Higher or Lower.

**Requirements 21:** On the internal page, under the “about the game” section for Higher or Lower there is another drop-down summary under that called “about the game” for the game – Quick RPG.

**Requirements 22:** On the internal page, under the “about the game” section for Quick RPG there is another drop-down summary under that called “about the game” for the game – Foal Shadow.

**Requirements 23:** On the internal page, under the “about the game” section for Foal Shadow there is another drop-down summary under that called “about the game” for the game – Spot the Difference.

**Requirements 24:** On the internal page, under the “about the game” section for Spot the Difference there is another drop-down summary under that called “about the game” for the game – Go with the BuffLow.

**Requirements 25:** On the internal page, under the “about the game” section for Go with the BuffLow there is another drop-down summary under that called “about the game” for the game – Sequential.

**Requirements 26:** On the internal page, under the “about the game” section for Sequential there is another drop-down summary under that called “about the game” for the game – Shopping List.

**Requirements 27:** On the internal page, under the “about the game” section for Shopping List there is a trademark for the game – Brainworks.

Results of our testing can be found in the Requirements.ods document on GitHub.

Upon designing the test cases for our website we learned that our website suffered from several oversights that we had committed. Our sign-up functionality would allow a user to sign up with no password and a username or vice versa and this was a serious infraction of how websites work. To correct this some error handling was added to the main.go file so that Username and Password cannot be empty strings. Testing our website also lead us to discover that some HTML features were not implemented properly. This resulted in an extra button to appear with a title on it that didn't originally look out of place but has since been remedied.

We also learned that browsers handle some of the HTML elements differently such as our summary element which appears with a drop-down arrow on Chrome but on Firefox it had no arrow but the functionality works.

However, we learned summaries are not supported by Microsoft Edge at all so that’s why we have fails in our test plan for Edge because the feature is not implemented properly in that browser. Certain features could not have a test case in our website, such as the time being displayed and our trademark but they are included in the requirements as parts of the website.

SYSTEM EVALUTATION CONTINUED

When comparing our app now to our original idea for this project our finished product is a lot more complex than what we first set out in the project specification. Originally, we decided on a Brain-Training app that would save scores to a database but over time we realized our app was getting cluttered with buttons and options so instead of overloading the user with too much information, and as mentioned in the website section we wanted a minimalist look so that any user could pick up our app and use it. So we as a team decided to separate out the high scores functionality and instead base a website around that part of the project. The UWP Application then sends the user to the hosted website using their default browser, whether it is on mobile or desktop.

As a result, our project specification grew in light of this decision. The website added a lot of complexity to our project as it needed to handle users logging in and signing up with our website as well as being able to display high scores achieved in our app. Therefore, when talking about if we achieved what we first set out in our project specification we, in a modest way, have gone above and beyond what we first set out not only in terms of size and scope but also in terms of complexity and functionality. We feel we have met the objectives of this project we set out to do and we are pleased with the result.

During the development of our app, we encountered some limitations with the technologies that we were using and that caused the team a lot of time loss. UWP caused us a lot of heartache when trying to connect it up to our MySql database as we knew we needed a Web API to do so. It took the team a considerable amount of time before we could even achieve “GET” requests to our database because of our Web API, SlashDB, not having a lot of documentation to understand. Once we eventually achieved a “GET” request we then set about getting a “POST” request to our database but we lost a lot of time on this part due to SlashDB’s documentation. As a result of this we had nothing to show at the end of it for a “POST” request. While we did have something to show for the “GET” request and included that in the application itself.

We decided the best route after all the time loss because of this limitation was to go with SQLite *(Appropriate Uses For SQLite*), this is the database most commonly used for high score saving in apps. This limitation worked out in our favour as MySQL is a safe secure database for handling users logging into and signing up to a website while SQLite is the better database for high scores as it meant even internet outages are accounted for and the score can still be uploaded later. SQLite also allowed us to use its in-built Swagger UI which saved us time with parsing the high scores into our website

Golang also caused the team some trouble as we learned through research and from among peers that Golang’s documentation is vague and hard to follow for people trying to learn the language. Since it is still relatively new, the official documentation is the only avenue for learning the language and then resorting to stack overflow after that to ask about more complex functionality. Stack Overflow allows us to get real-world examples of Golang in use which can be a lot more helpful than the Golang documentation which sets out information that wouldn’t be specific to your current situation. Luckily enough Swagger UI helped us considerably with the functionality we wanted to achieve in our website. However, due to how new Golang is we also had a tough time trying to host our website and also lost quite a bit of time on that part.

To ensure the application itself was tested we submitted the application to the Windows Store. Microsoft then emails a report on tests that were done so you can rectify any failed ones before uploading to the store. We also used the ‘Windows App Cert Kit’ which does the same thing automatically rather than waiting for an email. Below are the screenshots of status of the tests carried out. Unfortunately, Windows SDK for Windows Store Apps does not support the Sqlite.dll that we used within my application so we cannot submit it to the store. All other tests passed

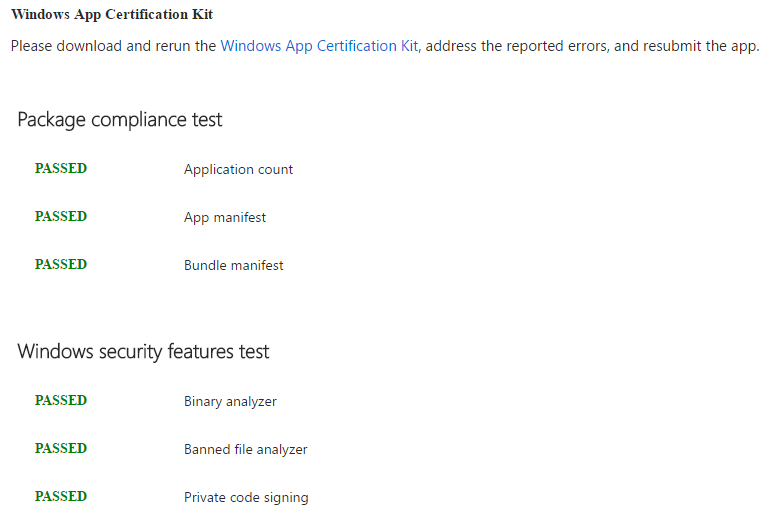


Figure - Microsoft App Certification 1

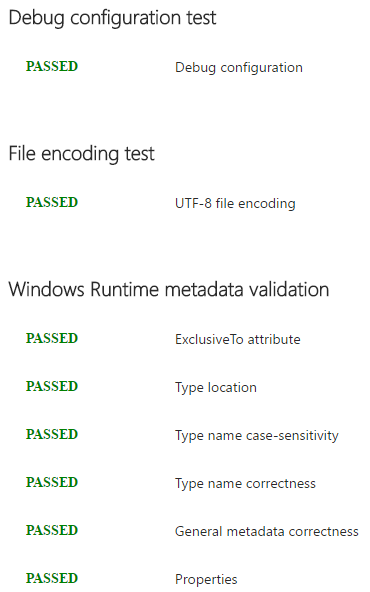


Figure - Microsoft App Certification 2

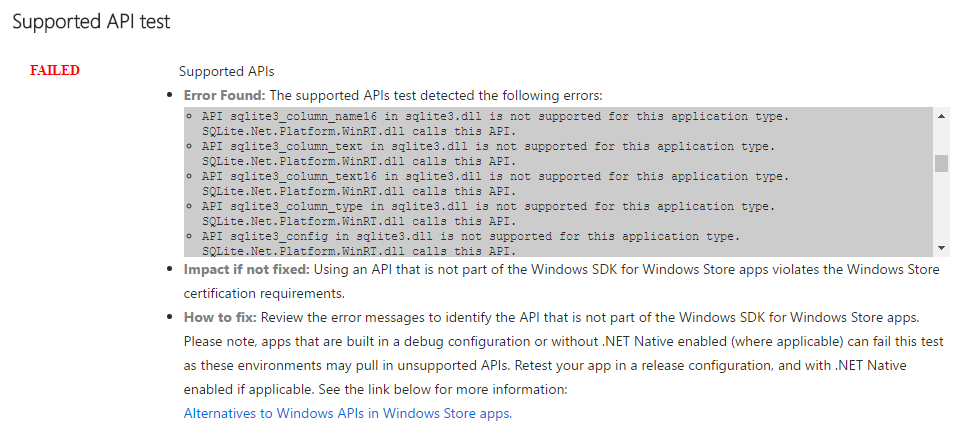


Figure - Microsoft App Certification 3

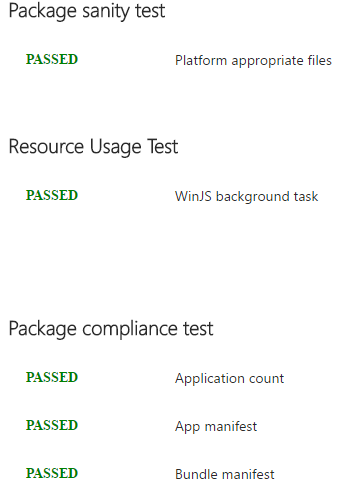


Figure - Microsoft App Certification 4

We had to delete the Bundle Artifacts and the App Package Upload folders that are created when you associate your app with the store in Visual studio on uploading the project to Github. This was because the project had exceeded GitHub’s maximum file size of 100MB. These folders caused the application to be over 3GB, which was much too large to upload.

CONCLUSION

In conclusion to our entire project, we can easily say this was the most interesting and thought-provoking project we have been given in the entirety of the four years we’ve had in college. Whereas most pieces of work we were given to complete is completed over only a few weeks, (a semester at most) this was to be continuously developed over the entirety of the academic year. Although we were tasked with a similar project to complete last year, that project was given a much smaller time frame to complete and the scope significantly reduced.

At the beginning, we had no idea how our app would turn out by the time the submission date came around. We wanted to create an application that would get the user to use their brain in a number of different ways with a set divide for the different style of games. This was eventually broken down into four categories; Attention, Problem-Solving, Memory and Speed. Each team member was given one to work with.

We basically wanted our application to connect to a database, which we eventually decided upon one on Azure, which would save the scores that users obtained from any of the games. So as not to unnecessarily overload the database with scores, we gave the player the option to decide whether or not the score they acquired would be saved.

To increase the scope of the project, we decided to incorporate a website. This would provide the user with more information about the games in the UWP with the help of summaries and display the scores that were achieved in the games. These functions are displayed once the user has successfully signed up and logged into the website, which was developed using Golang and SQL statements to search the database for the credentials that the user entered into the Signup and Login pages.

Golang caused us a lot of problems when we tried hosting our website online. Although Golang is a very efficient language to work with, there are not a lot of free services that support hosting websites that work from Golang. Our first two attempts at hosting did not work, as 000WebHost and Heroku failed to search the database, which was handled by the “main.go” file.

This resulted in these hosting sites instantly timing out as soon as we tried to sign up or log in to the website. Any attempt at fixing this did not seem to work, so we looked into hosting it on Azure, which we had already been using for databases and servers. This option seemed to run smoother as we eventually got the website up and fully operational on a Virtual Machine. The logging in and signing up problem still remained, except now it took a while to run and would return a timeout error. We later found out that it was the firewall causing the problem, and fixed it by configuring the inbound and outbound rules.

LEARNING OUTCOMES:

* Web Hosting - Hosted Golang website using Microsoft Azure Virtual Machine with IIS(Internet Information Service)
* UWP (Universal Windows Application) that incorporates SQLite for saving a user’s score and MySQL for displaying users of the hosted website
* Swagger UI [API] (explained in the UWP section) and SlashDB Web API (explained in same section)
* Improved Team Management due to working together on a project for an extended period of time
* Experience gained in working on a large-scale project. This to-date has been the largest project we’ve had to work on in the last four years in college
* Communication has improved, due to emailing Supervisors and contacting team members whenever major problem arose
* Team Management – A few days each week, we made arrangements to book out rooms in the library for a couple of hours to work together on the project, and talk out any ideas or problems we may have had
* We worked in an agile fashion – This is the preferred methodology for most industries, which gives us the valuable experience we need for working in industry.

FUTURE INVESTIGATION:

* SQLite3.dll - When using SQLite for our database we discovered that you cannot pass the ‘Supported API’ test when running tests using the Windows App Cert Kit while this .dll is included. While unfortunate, there still may be a work-around for this. The App Cert Kit allows you to test your C# application and it also provides the user with some helpful tips on how to fix the issue. While this is helpful, we can only really test the finished application and by the time we had associated the application with the Window’s app Store we didn’t have a lot of time before the deadline to further investigate the issue. This could be something to work on at a later stage if the project was revisited
* If we were to continue working on this, we would spend more time working on the look and feel of the app. We would do this by implementing transitions between app pages, and improve more of the graphics from the application and website. As developers, it is well known that we are not design-focused, we are mainly back-end functionality focused
* Have the games ported over using JavaScript – One suggestion from the supervisors was to incorporate the games from the app on the website, using JavaScript. This proved to be very difficult to implement with the time we were given, but with more time, could be implemented and allow for more people to access the games.
* Include more analytics into the application/website – Our website and app only shows information on the users who have registered to the website and any scores that were saved on the UWP side. If given more time, we would include more information, such as how long the user had been a member for and how much time the user spent playing each of the games.
* Implement multiplayer functionality – If we had more time to work on this project, we would try to create more games. These would allow multiple players from separate systems to play against each other, and the winner(s) would be decided by who had the highest scores or survived the longest etc.
* We had thoughts about transitioning over to a different IDE such as Unity, as it has pages upon pages of documentation for just about every one of its built-in functionalities. We had decided not to use it from the beginning as, again, we weren’t design-focused at the time and instead wanted to focus on honing our C#/UWP skills. If we continued working on this project, we would make plans to port it over to Unity. This could then be uploaded to sites such as GameJolt and Steam to get feedback from a larger audience. We would be able to include Beta testing for our application this way.
* If more emphasis was placed on distributing the application on multiple platforms, we could implement Xamarin or Angular2/Ionic2. This would allow for non-Windows devices such as Android and IOS to run the application with minimal tweaking to the codebase
* If we had been more design-focused with the application, we would’ve added a feature for when a user signs in with their username and password. Then the username they signed up with could be displayed on the Internal website page and the UWP application to increase the n-tier feel. This would make the website feel a bit more personal.

Overall, it was a very challenging but rewarding project. We implemented a lot of what we were taught by our lecturers with just as much knowledge that we learned from our own study.

Although there are a few things we would change if we were to do it again, we are immensely proud of what we have accomplished.

We are particularly proud of the amount of work we got done, considering the number of modules we had to continuously study for and submit projects in while also trying to get weekly work done on our Dissertation.

REFERENCES:

Retrieved from Brain HQ: http://www.brainhq.com/

Retrieved from SlashDB: https://www.slashdb.com/

Retrieved from Hurl.it: https://www.hurl.it/

Retrieved from Imgur: http://imgur.com/

Retrieved from Cmder: http://cmder.net/

*Retrieve data from URL into JSON*. (n.d.). Retrieved from DaniWeb: https://www.daniweb.com/programming/software-development/threads/475686/how-to-retrieve-data-from-url-into-string-json

*What's a Universal Windows Platform (UWP) app?* (2017, February 8). Retrieved from Microsoft: https://docs.microsoft.com/en-us/windows/uwp/get-started/whats-a-uwp

*Agile-101*. (n.d.). Retrieved from VersionOne: https://www.versionone.com/agile-101/

*Appropriate Uses For SQLite*. (n.d.). Retrieved from sqlite: https://sqlite.org/whentouse.html

*Asp.NET Web API - 405*. (n.d.). Retrieved from StackOverflow: http://stackoverflow.com/questions/9854602/asp-net-web-api-405-http-verb-used-to-access-this-page-is-not-allowed-how

*Bcrypt*. (n.d.). Retrieved from Sourceforge: http://bcrypt.sourceforge.net/

*CLI*. (n.d.). Retrieved from searchmicroservices: http://searchmicroservices.techtarget.com/definition/Common-Language-Infrastructure-CLI

*Cloud Application Platform*. (n.d.). Retrieved from Heroku: https://www.heroku.com/

Daniel J. Simons, W. R. (2016). Do “Brain-Training” Programs Work?

*Displaying a part of the JSON returned from a DB in UWP*. (n.d.). Retrieved from StackOverflow: http://stackoverflow.com/questions/42559804/displaying-a-part-of-the-json-returned-from-a-db-in-uwp

*Error on getting one part of JSON* . (n.d.). Retrieved from StackOverflow: http://stackoverflow.com/questions/42564517/error-on-getting-one-part-of-a-json-response-message-to-display-in-a-textbox

Evan Comen, M. B. (2016, April 15). The Cost of a Computer the Year You Were Born.

*Facebook Developer*. (n.d.). Retrieved from Facebook: https://developers.facebook.com/)

Finley, K. (2012, July 14). *What Exactly Is GitHub Anyway?* Retrieved from Tech Crunch: https://techcrunch.com/2012/07/14/what-exactly-is-github-anyway/

Flanagan, D. (n.d.). *JavaScript - The Definitive Guide.* O'Reilly.

*Free Web Hosting with PHP,MYSQL and cPanel*. (n.d.). Retrieved from 000webhost: https://www.000webhost.com/

*Git*. (n.d.). Retrieved from https://git-scm.com/

*GitHub*. (n.d.). Retrieved from GitHub: https://github.com/

*GitHub*. (n.d.). Retrieved from https://github.com/

*How can I change the default website connection timeout in Chrome?* (n.d.). Retrieved from superuser: https://superuser.com/questions/633648/how-can-i-change-the-default-website-connection-timeout-in-chrome

*How to put a text beside the image (html)?* (n.d.). Retrieved from StackOverflow: http://stackoverflow.com/questions/19302122/how-to-put-a-text-beside-the-image-html

*ISO/IEC 23270:2006*. (n.d.). Retrieved from www.ISO.org: https://www.iso.org/standard/42926.html

*Json Serialization and DeSerialization*. (n.d.). Retrieved from YouTube: https://www.youtube.com/watch?v=76blDatESaA

*Lumosity*. (n.d.). Retrieved from https://www.lumosity.com/

*Microsoft Azure*. (n.d.). Retrieved from https://azure.microsoft.com/en-us/

*Microsoft Blend for Visual Studio 2015*. (n.d.). Retrieved from https://social.technet.microsoft.com/wiki/contents/articles/32741.microsoft-blend-for-visual-studio-uwp-creating-a-simple-animation.aspx

*Middleware Solutions for the Enterprise*. (n.d.). Retrieved from microsoft: https://www.microsoft.com/africa/enterprise/middleware.aspx

*Middleware Solutions for the Enterprise*. (n.d.). Retrieved from https://www.microsoft.com/africa/enterprise/middleware.aspx

*MySql Workbench*. (n.d.). Retrieved from MySql: https://www.mysql.com/products/workbench/

Nadia, S. (1993, December 15). KID'S BRAIN POWER.

*N-Tier Data Applications Overview*. (n.d.). Retrieved from msdn.microsoft: https://msdn.microsoft.com/en-us/library/bb384398.aspx

Piltch, A. (2016, May 12). PCs Aren't Dying. They're Just Way Overpriced.

Plain\_Vanilla. (n.d.). Retrieved from QuizUp: https://www.quizup.com/en

*Return JSON file with ASP.NET Web API*. (n.d.). Retrieved from StackOverflow: http://stackoverflow.com/questions/15725884/return-json-file-with-asp-net-web-api

*SDLC-RAD Model*. (n.d.). Retrieved from tutorialspoint: https://www.tutorialspoint.com/sdlc/sdlc\_rad\_model.htm

*Selenium Tutorial: Project*. (n.d.). Retrieved from newcircle: https://newcircle.com/bookshelf/selenium\_tutorial/project

*SlashDB*. (n.d.). Retrieved from https://www.slashdb.com/

*SlashDB Documentation*. (n.d.). Retrieved from https://www.slashdb.com/documentation/

*SO\_Questions*. (n.d.). Retrieved from StackOverflow: http://stackoverflow.com/questions/42211797/trying-out-a-tutorial-for-getting-json-data-from-a-web-api-and-i-stumbled-on-thi

*SQLite Overview*. (n.d.). Retrieved from https://www.tutorialspoint.com/sqlite/sqlite\_overview.htm

*Stack Overflow Developer Survey 2016*. (n.d.). Retrieved from Stack Overflow: https://stackoverflow.com/research/developer-survey-2016#technology-desktop-operating-system

*Standard ECMA-334*. (n.d.). Retrieved from ECMA: https://www.ecma-international.org/publications/standards/Ecma-334.htm

*The Go Programming Language*. (n.d.). Retrieved from https://golang.org/

*The Official Microsoft IIS Site* . (n.d.). Retrieved from IIS: https://www.iis.net/

*Top Programming Languages Used in Web Development*. (n.d.). Retrieved from cleverism: https://www.cleverism.com/programming-languages-web-development/

*Trying to return users from a database in a textbox*. (n.d.). Retrieved from CodeProject: https://www.codeproject.com/Questions/1174816/Trying-to-return-users-from-a-database-in-a-textbo

Waldo, J. (2006). On System Design.

*What are basic differences between IaaS, PaaS and SaaS?* (n.d.). Retrieved from Quora: https://www.quora.com/What-are-basic-differences-between-IaaS-PaaS-and-SaaS

*What is a DLL?* (n.d.). Retrieved from www.microsoft.com: https://support.microsoft.com/en-us/help/815065/what-is-a-dll

*What is Coupling?* (n.d.). Retrieved from TechTarget: http://whatis.techtarget.com/definition/coupling

*What is Platform Agnostic?* (n.d.). Retrieved from technopedia: https://www.techopedia.com/definition/23666/platform-agnostic

*What is Scrum?* (n.d.). Retrieved from https://www.scrum.org/resources/what-is-scrum