CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

A Secure and Serverless Approach to Verification of Student Records

A graduate project submitted in partial fulfillment of the requirements

For the degree of Master of Science in Computer Science

By

Karandeep Singh Bhamra

August 2019

The thesis of Karandeep Singh Bhamra is approved:

Li Liu, Ph.D.	Date
Robert McIlhenny, Ph.D.	Date
Taehyung (George) Wang, Ph.D., Chair	Date

California State University, Northridge

Acknowledgements

I am grateful to Dr. George Wang, for being my committee chair and giving me a chance to work on this project. In addition, I would like to express my gratitude to Dr. Li Liu, and Dr. Robert McIlhenny for agreeing to be the additional members of my committee.

A special thanks goes out to my wife, Jessica, and my parents. The support my family provided me with went a long way in helping me finish this project.

Also, I would like to thank my friend Jesus Moran Perez, he was the one who initially inspired me with the idea for the project and has been of great assistance in helping me complete this project.

Table of Contents

List of Figures

Abstract

A Secure and Serverless Approach to Verification of Student Records

By

Karandeep Singh Bhamra

Master of Science in Computer Science

Most academic institutions do not self-operate the system that is used for the verification of student records. Such a task is contracted out to firms who act as the middleman between the student and the potential employer who might be seeking verification of records. This project proposes a complete overhaul of that system and introduces a fast, and secure alternative system which accomplishes just that while maintaining low operating costs by making use of serverless architecture. The project consists of a scalable frontend website which is used to request a student's record, the request is then processed by a web framework which routes it to an API endpoint where a serverless function retrieves the student's record from a scalable database and generates a verified pdf certificate which is sent back to the frontend website client to be displayed. The project also implements tools for administering the blockchain like data structure that holds the student's records and for verifying the integrity of the student records. The completed program demonstrates the entire workflow from the viewpoint of the client and the school administrator, while showcasing the benefits of implementing the system in a serverless architecture and how the student records are safe guarded against malicious modification.

Chapter 1: Introduction

Verifying a degree or any student record takes some amount of time, capital and patience. In addition, the verification process requires potential employers and background check firms to request confirmation of credentials from third party services associated with the academic institution rather than the institution directly. The results are not immediate and the privacy is implied, but not explicit. The data transaction should exist solely between the student, the institution and those that you want to share it with. A third party cannot be completely trusted to handle your private data. On the opposite spectrum, the student cannot be completely trusted to provide accurate results, as the student providing it can manipulate it before delivering the modified records to those who requested it.

Since it has been established that verification of academic records takes time and money, most academic institutions have contracts with firms that provide the academic record verifications. There is unnecessary overhead related to that process; an employer or background check firm usually has to make a request with those middleman firms, who will then forward the request for records to the institution itself, or fetch the information from their own database which was provided to them [1]. I propose a solution in which the middleman can be cut out, and the verification process is provided directly by the academic institution. Overall, the main purpose of this project will be to create a proof of concept data storage format that borrows from and mirrors the blockchain data structure. In said data structure, the blockchain will hold a growing list of academic records of students, and will allow for the verification of the authenticity of those academic records. Once the format has been implemented, it will then be utilized to

show how an academic institution can keep accurate and verifiable records relating to the students, and how a third party might be able to request and receive those records on an infinitely scalable serverless architecture.

Serverless architecture is a relatively modern concept. It is an application design where the business does not have to worry about managing the infrastructure that would support their application. In serverless architecture, there are still servers somewhere in the backend, however, the customer of those servers, is not responsible for managing them in any way. A person, business or any client would not have to worry about having to update the server, software, libraries, or be required to hire employees that would do the managing; instead, cloud providers such as Microsoft, Google, or Amazon, provide and manage the servers on your behalf, thus providing a seamless experience to the end client [2]. Serverless architecture relies heavily on event-driven programming, and executes code without provisioning servers through the use of functions. Functions are the pieces of code that make up the business logic, and they can be run within few milliseconds and the entire functions platform is provided as a service by most cloud vendors [3].

The term blockchain has taken off in popularity due to its connection to cryptocurrency. However, the data structure known as blockchain has many uses besides just cryptocurrency. A blockchain is a construct that acts as a digital database where the transactions are recorded in a secure and public manner. The concept can be simplified to this: anything can be owned, and there are things that we own and would like to share those items, and blockchains act as the middlemen which facilitate said exchange. A block refers to a specific record, or data that can be permanently recorded. A blockchain

is a collection of those blocks appended one after another, and the blockchain acts as a historical record of all the total blocks added to date [4].

In a blockchain, the initial block is also known as the genesis block and is always hardcoded into the application. Once a block has been added to the blockchain, it is difficult to change that record [5]. The records are also verified by the network to make sure that they are valid before they are added to the blockchain. Each block also contains a unique code called a hash, and in addition, the current block also contains the hash of the previous block in the blockchain. A hash code is an output of a mathematical function, which is generally a string, that is composed up of series of characters and numbers. So, for a given unique input, the hash function will always yield the same hash; therefore, for someone to modify the blockchain, they will have to compute hash codes for each block and that is an extremely computationally intensive task depending on the underlying hash implementation. The benefit of a data structure like blockchain is that it is a digital log of all transactions that have occurred and due to its nature, it can be quickly searched and verified.

This project is organized in four distinct parts: serverless backend, the serverless frontend, serverless database, and an administrative student record software. The backend portion relates to the serverless computing provided via functions, which are ultimately responsible for providing the frontend client with the document that was verified and generated on the fly and transmitted over a secure communication channel. The frontend will be a website where the client enters the student information such as name and then the request is routed via a web framework to the corresponding serverless function and the received document is then displayed if the student record exists. The database is

necessary for the storage of student records, the focus of the database will be on making it scalable, affordable and integrating the blockchain like data structure with the serverless functions, which when used in cryptocurrency, it is a decentralized system due to peer to peer network; however, for my project it will be a centralized system controlled by the institution due to an easier integration with the statefulness of the project. Lastly, since the project is a complete overhaul of how the records are handled now, it was necessary to showcase how an administrator in charge of this system would utilize it. The tool would allow a person in charge to load student records in a predefined file format, create a new student record, save the student record in the database, and verify the integrity of the records in the blockchain.

For a complete understanding of this project, it will be necessary to discuss other advancements in the area of using blockchains for academic institutions and related works, the underlying technology and algorithms used in serverless applications and blockchains. In addition, we will look at and compare the benefits of serverless versus other traditional architectures, the implementation of the project and its subcomponents, the workflow of the whole system, the testability of the project and any further improvements that could have streamlined the whole operation if not for the time or resource constraints.

Chapter 2: Related Work

As of the writing of this thesis, there are multiple corporations and startups that are starting to look towards blockchain to solve a host of problems. The storage and verification of data is a huge section of the outstanding problems left to be solved. Most recently, the popular hardware and software company IBM has partnered with Sony to make use of blockchain technology. IBM has created their own blockchain called IBM Blockchain, which will be the basis for a new platform geared towards academic institutions [6]. Their goal is to allow students and schools to track each other's progress. Additionally, such a system will make use of the built-in transparency and accountability due to it being based off of blockchain technology. According to IBM Japan, there are many preparatory schools based around Japan that have already incorporated IBM Blockchain along with student credentials to keep track of coursework and verify transcripts [7]. A fully managed cloud service such as IBM Blockchain allows for institutions to rapidly adopt the technology without having to pay for new research and development for a private blockchain network.

Learning Machine is another small, private company which is hoping to make a big splash in the blockchain scene [8]. Their goal is to provide a complete system to public and private bodies which will allow those parties to issue official, instantly verifiable records anywhere in the world. From the examples shown on their website, they issue verification for any type of company-wide or institution wide record. They have examples of an individual's doctorate degree being verified and a certificate is issued with their information and the date their degree was conferred upon them. They intend to pioneer the way to allow businesses or schools to easily issue instantly verifiable records.

Chapter 3: Serverless Architecture

The tech industry is plagued with buzz words and new ideas that seem to be offshoots of old ideas. Most recently, “serverless computing” (also known as serverless architecture or function as a service, FaaS) is an application design pattern where cloud providers manage the infrastructure and your applications simply build and run on that infrastructure. At a glance, this is a great way to create and deploy applications. It allows for the client of the serverless architecture to focus on their business logic, without having to worry about the hardware, scaling and maintenance aspect of servers. The term serverless is however misleading. There are still servers, because otherwise it would hard for code to run on nothing, but, the person or business that is creating or deploying the application does not have to provision any servers personally [9].

3.1 Serverless History

Computers were not as powerful or portable as they are now. With the emergence of personal computers, much of computers moved away from the large, centralized mainframe systems to a smaller client-server model. Early on, when a client wanted to deploy an application, they would have to provision their own physical, bare metal server, install the required software tools, language frameworks, and any other dependencies of the application. In the beginning, all servers were bare-metal servers, which meant that each individual, physical server used by a single client [10]. They would require a dedicated person of expertise whose sole job was to administer the server, and would be known as Server Administrators. The need for such a person was an additional cost for the client of the server, in addition to having to pay for any resources

related to the cost of operation such as electricity, and other on-premise infrastructure costs. Eventually, with the development of virtualization, it became easier to rent out the server to multiple tenants. Now, a single machine could host multiple websites. Software and other web applications could be run on these virtual machines with completely different operating systems. It now became possible to run an application that depended on the Linux operating system and at the same time run another application that depended on the Windows operating system. Even with virtualization, there were still costs associated with it. The monetary costs dealt with any software or operating system licensing fees. Overall, virtualization led to a decrease in general costs due to the idea of multiple tenancy [11]. A virtual machine could be created with specified number of virtual processors, dedicated memory and storage space; so, an application that required fewer resources would not be charged the costs associated with the higher resources. It became cheaper to run smaller applications rather than larger, resource hungry monolithic applications. One negative aspect of virtual computing would be that since the resources were divided among the various virtual machines, a larger, resource hungry application could easily starve out other applications. There are also issues with scalability, if your software becomes too large too quickly, the set, finite resources might struggle to keep up with the popularity. Bare metal servers and virtual machines still have their place in computing to this day and age, however, they did lead to newer, and improved ideas.

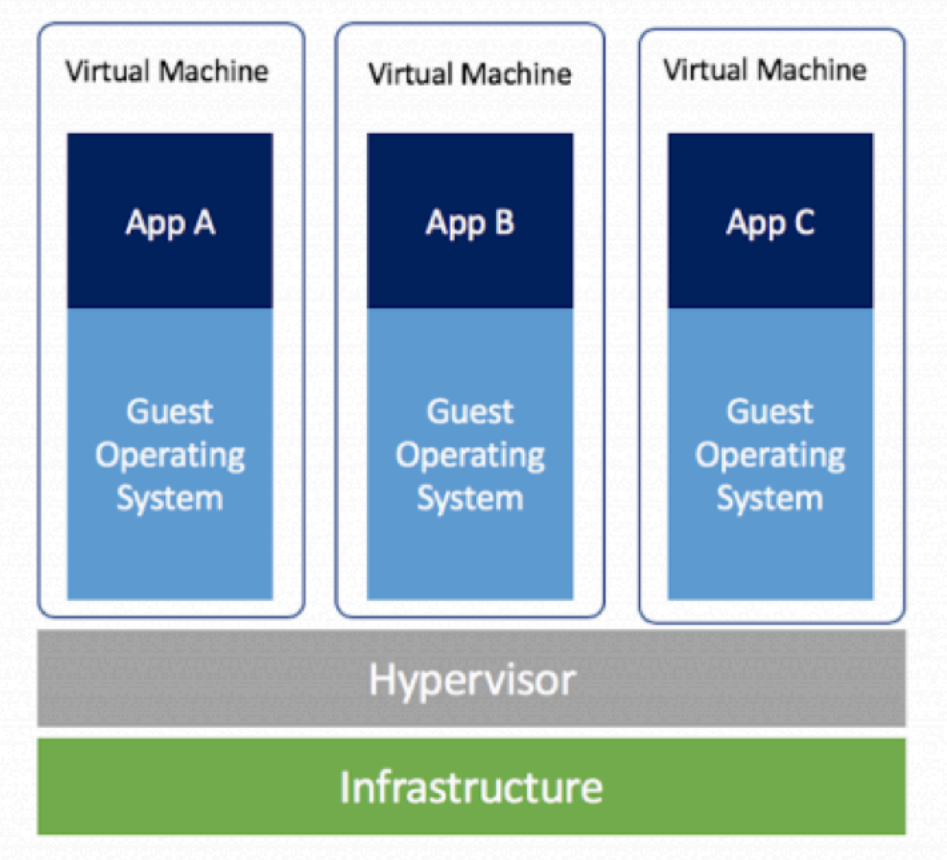

Figure 1 Virtual Machine

After virtualization, the next big idea was containerization. Application containerization is still a type of virtualization, but, a container consists of an entire runtime environment. In essence, you are abstracting away the differences of various operating systems and hardware infrastructure, by bundling the application with all of its dependencies and files needed to run it into a single package. Docker is one of the most popular software that performs the operating system level virtualization [12]. A single application can be packaged into a single docker container image which can be downloaded by anyone and installed on any operating system. It is now possible to run applications specific to a single operating system on any operating system platform that has the Docker software. Benefits of containers include consistency when shipping

applications, you know the application will behave in a deterministic manner since it was tested and developed in a controlled environment. As popular as Docker is, it is not a one size fits all solution to computing. Some disadvantages of containers are that not all applications benefit from containerization, and there is a speed penalty when running a container. In virtualization or containerization, there is a performance overhead; containers are faster than virtual machines, but still not as fast as bare metal. Lastly, the data inside of a container is not persistent [11]. Due to its design, the data generated when the container is spun up is lost when it is shut down, however, there are ways to save data from a container.

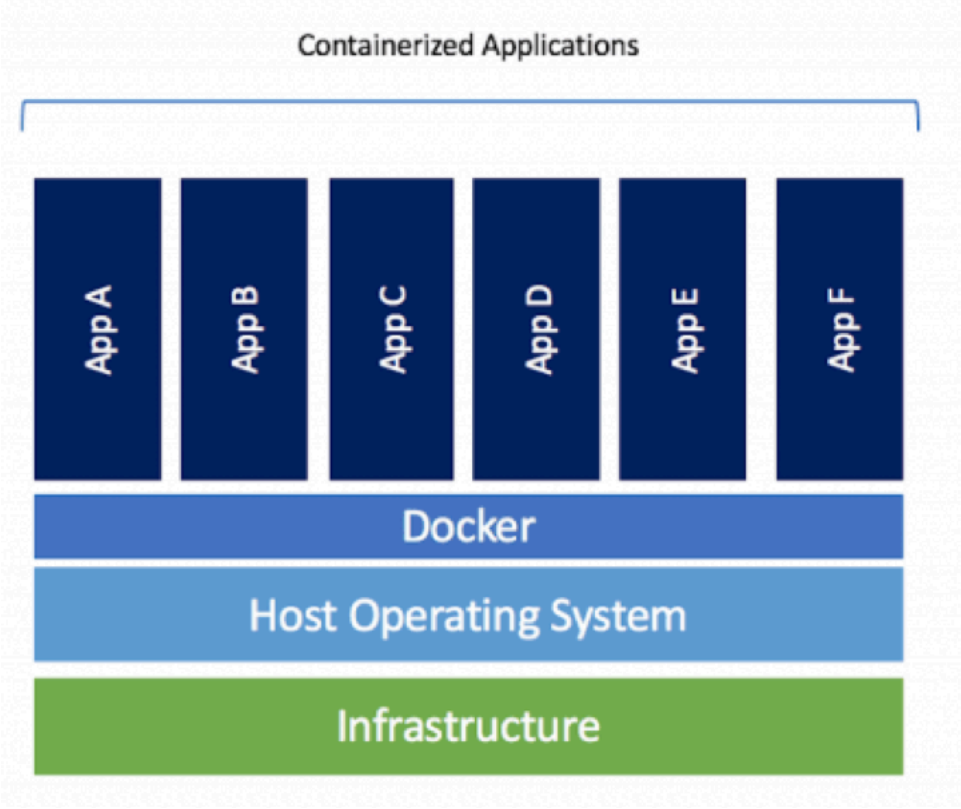

Figure 2 Containerized Applications

The latest fixation brings us to now, which is serverless computing via functions as a service. Functions are useful since they provide an on-demand functionality. Functions are a finite piece of code which are run when they are invoked. It is easy to become familiarized with functions since their underlying technology is based on containers. Functions are event-driven and are therefore invoked by any number of events, upon being invoked, the container where the code resides is spun up and the code is executed and the container is shut down [2]. Therefore, you only pay for the amount of time your code took to execute and not for any of the underlying infrastructure. All function as a service provider also allow for easy and quick scalability of resources. When a function needs to scale up to meet a million requests or scale down to handle a single request, the cloud vendor handles all the underlying allocation and provisioning of resources. Since a function is just a program that will be executed, most vendors allow for programming in plethora of programming languages and have support for many of the popular frameworks. The usefulness of a function is defined by having it connect to various other cloud services such as databases.

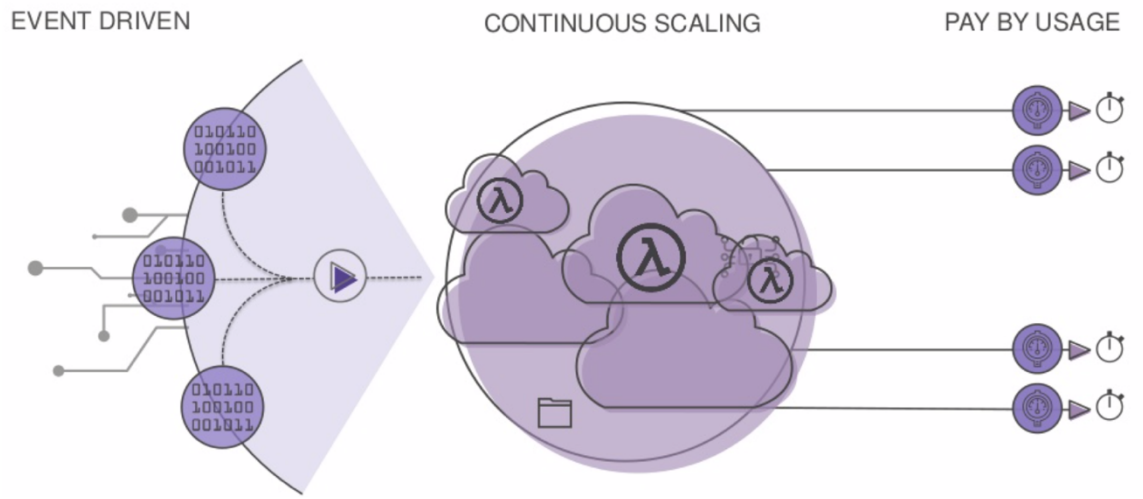

Figure 3 Event-Based Functions

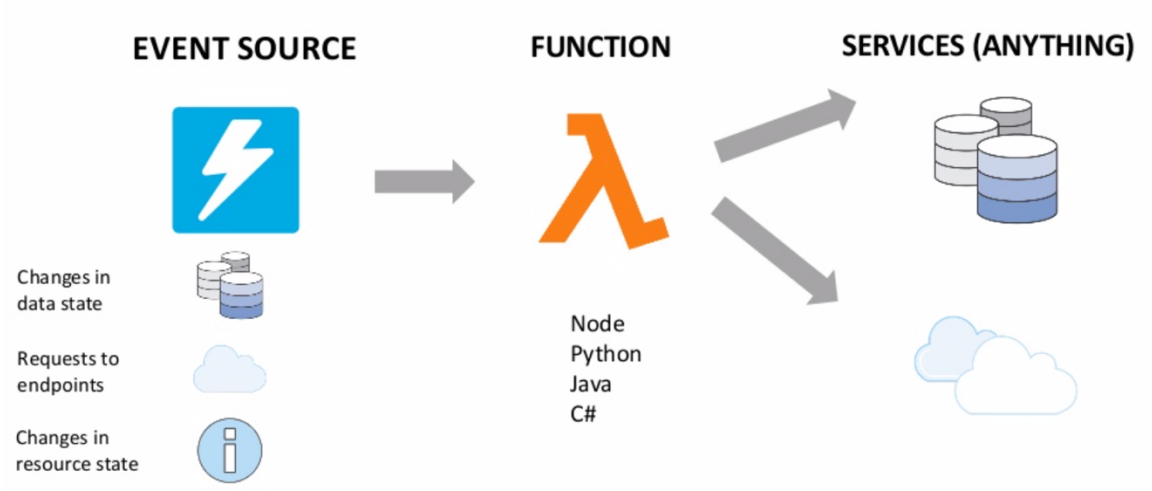

Figure 4 Function Integration

3.2 Serverless Platform

Functions are not the only cloud computing service that makes up a serverless architecture. As previously discussed, databases are also provided as a scalable service. Manually having to manage the capacity of a database can be time consuming; it is difficult to predict when there might be a spike of incoming or outgoing requests, so you have to be able to handle those requests dynamically. Many cloud services providers now have serverless databases that are simple, scalable, cost effective and highly available. Availability is one of the most important attributes associated with databases. If a database is unavailable, then the users of its data such as clients, and applications are unable to access it. Having an unavailable database is a detrimental to anyone relying on them. Due to the takeover of responsibility by the provider, developers and operators can save time and money by focusing on their business logic part of the application and less dependent on infrastructure support teams. In addition to databases, serverless platform also depends on managed services for API endpoints, message queueing service,

continuous integration and continuous delivery pipelines, user management, and storage. Just about every important part of computing can be extracted to a managed service and offered to clients in an easy to use manner.

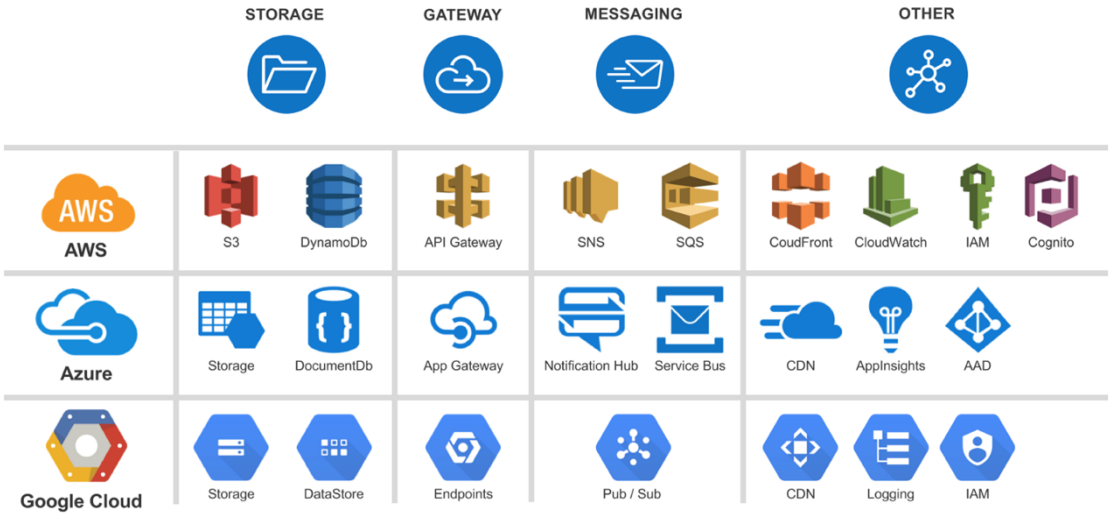

Figure 5 Serverless Platform

Chapter 4: Underlying Project Technology

Due to this project's reliance on the serverless architecture and concepts, there are a few technologies that are used heavily. For starter, this project made use of serverless functions as the compute engine, because there was a need to execute code as required by the workflow of the project. The serverless functions helped facilitate the logic required to add student information in to the database, to query the database, and to generate the documents which will be returned to the end user and much more. In addition to compute functions, another technology that was essential to the project was a database. Almost every single program that needs data to persist after a power cycle has to make use of a storage medium one way or another; it is literally quite impossible for data to exist unless its stored in a medium somewhere such as memory, files or a database. This project needed a database which would act as the basis of the blockchain like structure. The requirements for the structure mandated that the database be scalable. Scalability was the primary factor, because it needs to be able to handle a dozen requests to multimillion requests, even though in reality it might never actually quite reach that limit. However, the application needs to be highly responsive and always online, so the compute functions and the scalable database were essential. In addition to the components that make up the serverless architecture, blockchains also rely heavily on concepts from cryptography such as hashing via hash functions and a related topic of collisions. The next few sections will dive deeper into the listed topics from above and we will see how they played a vital role throughout the project.

4.1 Serverless Functions

Serverless functions refer to pieces of relatively small code which will get executed in a completely managed environment. Serverless functions can be a valid tool in most applications. It is hard to advocate for them for every application or role, but they have their uses and the persons deploying them has to think and play to their strengths. The strengths of functions are that they are relatively easy to set them up and get them running code you might have. Most cloud vendors have their own implementation of how they interact with their cloud services and most of the cloud vendors have the same relative restrictions on the functions. As opposed to full on virtual machines or servers with language software development kits (SDK) installed and almost no limitations on how the programming language can interact with the filesystem via its application programming interface; functions are limited on which service they can access with the provided permission roles and how long they can execute for, and how much memory they are able to occupy and make use of. However, at the same time, they also benefit from significantly reduced operational costs, complexity and engineering lead time [2].

Serverless functions can be as small as few lines of code which can be edited directly in the cloud providers web interface and can be deployed directly to live sources from the online interface. It is however not advisable to perform all the programming or testing from the web interface. The difficulty comes in that such interfaces are often quite clunky and different that how a person, a team or company prefer them. For example, Microsoft Azure Functions can be edited and deployed directly from their dedicated Azure Functions webpage. In addition, one is able to write code directly via the available online code editor; if online editing is too cumbersome, you are able to write code offline

on your device and then upload the project and all its files via the online upload files options. Even if that is not up to the developer's liking, they are able to use Microsoft's development tools such as Visual Studio or their text editor Visual Studio Code which have support via officially released plugins which allow you to create functions, write code and then upload it from the software. In essence, Microsoft provides flexibility and choices to the developers. On a personal experience, I have found their tools and support to very helpful and friendly towards developers.

Most cloud vendors such as Google, Microsoft and Amazon also support multiple programming languages for their serverless compute functions. For instance, Microsoft currently supports JavaScript, Java, Python and their programming language C# [13]. On the other hand, Amazon supports JavaScript, C#, Java, Go, Python, Ruby, PowerShell and also provides a Runtime API which allows developers to use any additional languages to create the functions [14]. When choosing a cloud vendor, a team has to be aware of all the services and languages that are supported for that given function provider. On a personal note, it is also advisable to see if the cloud provider provides the developer with an offline SDK which can be used to test the function before deploying it to a live resource. Since the cost of the function is computed when it is run and the calculation incorporates variables such as the execution time, the amount of ram used and if it interacted with any other services, it is useful to test it on a development environment to make sure that the team is not being charged for testing while developing.

On an operational level, it is also important to discuss how a serverless function is executed. Generally, when a team has decided on developing their software with functions in mind, they are also going to be making use of other cloud services such as an

API gateway or a database. There is generally a single way that a serverless function can be executed, and it is via some received event. The receiving event can be an HTTP method such as GET or POST. The Hypertext Transfer Protocol (HTTP) is used to communicate between a client and a server, it is a request-response protocol [15]. An HTTP request can be made to the web address that is exposed for the serverless function, which will in turn raise an event which is intercepted and is responsible for the function executing either based on the request payment or other determined workflow. The functions underneath is run on container technology, meaning that when the request is received, the container where the code for the function is hosted, is spun up and the code is then run and the output is logged and sent to wherever specified and the container is then shut down. This is the biggest drawing point of the serverless architecture, you only pay for the code for as long as it ran and the resources it used. The container does not exist for much longer than the time taken to execute the code. The concept of just having your code run and only worrying about the business logic, and having the rest be fully managed is a refreshing idea.

4.2 Serverless Databases

In a serverless architecture, it is important that as many possible components or services be serverless as well. The main goal is to be highly scalable and highly available. It does not matter if the application is ground breaking or the next greatest thing, if no one is able to access the application due to it being overloaded, it will never be popular. The application should be accessible from a dozen clients to a million clients with no perceivable degradation. A database is generally the most important part of the workflow of any given software. The database is responsible for storing information that is going to be used in the business logic, so it needs to be available at any given time. Generally speaking, a database is usually installed on a server and the database users are created and the database management system is the middle layer which interacts between the data and clients.

Databases themselves come in a variety of flavors and support different model system. The two most popular database models are relational and document based. The relational database model supports collections of items with set, pre-defined relationships between them. Relational databases also support a single unified query language that is used to interact with the database and the data, it is called the Structured Query Language (SQL). SQL was adopted in the 1980s as the industry standard [16]. For relational databases, such a standardization led to its huge growth in popularity, one only had to learn a general form of the SQL standard and suddenly they had the knowledge to work with almost all the SQL databases. Not every single database has the same SQL language implemented, there tend to small differences from one to another. That's why the standard was created and even then, not every vendor supports the whole standard. SQL

works well with databases with relations between the objects. In a database, there is a table which consists of rows, each row can have multiple columns for various data required. Tables are the main components of a database, since that is where the data is stored. A row can have one or more columns of various datatypes. For example, there can be a table called Users where each key can be a unique user and the column for that row can have information such as first name, last name, and age. Every column must have a datatype, and it is up to the developer to find the best datatype that fits the information that will be going into the column [16]. SQL databases are by far the most popular these days and they have remained so due to nature of the data that the application is manipulating.

The other popular type of databases is referred to as NoSQL databases. NoSQL which stands for "no SQL" or "non-relational" is an alternative which provides ways around some of the pitfalls of the relational databases. They store and manage data in such a way that allows for developers to optimize their applications for high speed and extreme flexibility. NoSQL databases were pioneered by some of the giants in the tech field such as Google, Amazon and Facebook. The biggest benefit of NoSQL databases is that it can be scaled horizontally across plethora of servers, which a SQL database cannot accomplish in an easy manner [17]. There are plenty of disadvantages to NoSQL as well; firstly, if the data is highly relational, it becomes harder to represent in a non-relational database schema. Additionally, NoSQL does not provide the same level of data consistency as a traditional SQL database [17]. This is primarily because where the traditional SQL databases focus on the reliability of data transactions, the NoSQL sacrifices that aspect to focus on speed and scalability. One additional benefit of NoSQL

is also that any given data can be stored in any other record. There are four popular types of NoSQL databases, and they are document databases, key-value stores, wide column stores and graph databases. We will be focusing on the first one, document database.

This project needed a scalable database and the data for each student was already a bit based on a "page" of the student's record. So, it made sense to choose a document-based NoSQL database such as Microsoft's Azure Cosmos DB. Azure Cosmos DB is Microsofts answer to a globally distributed, scalable database as a service. It allows users to scale the throughput required and storage across their highly available Azure regions worldwide. Additionally, when designing the project, I did not have to worry about a database schema. Designing a database schema can be time consuming in itself, so, when you can just store data as a JSON formatted page, it allows one to focus on other aspects of the project. By far the biggest benefit of using Azure Cosmos DB was that it had support for a range of APIs for querying the database [18]. Since I was already familiar with the SQL style of querying databases, it was easy to use the SQL API to query my document base NoSQL database. For someone that might not have the familiarity with a NoSQL type database, Cosmos DB is an excellent entry point due to its familiarity for those with SQL experience.

Sample JSON Schema

```
{
        "title": "Example Schema",
        "type": "object",
        "properties": {
                "firstName": {
                        "type": "string"
                },
                "lastName": {
                        "type": "string"
                },
                "age": {
                        "description": "Age in years",
                        "type": "integer",
                        "minimum": 0
                }
        },
        "required": ["firstName", "lastName"]
}
```

Figure 6 JSON Schema

4.3 Hash Functions

Cryptography is used heavily throughout blockchains due to their nature of being cryptographically secure. Even though my project borrowed heavily from blockchain without actually being a blockchain per se, I still needed to incorporate hashing and hash functions into the project. Hashing is the process of making use of a hash function to generate a constant output for a given unique input. So, a hash function takes in an input, generally a string of any length, and it maps the input to an output which is a value of a specific length [19]. In the context of blockchains in cryptocurrencies, the transactions on the blockchains are taken as an input and run through a hashing algorithm which returns an output. So why is hashing required and important? For example, if we look at a popular hashing algorithm such as SHA-512, we know that our output will always have fixed length of 512 bits. This allows the input of any given length to always result in a hash with its length being 512 bits long. There are a few properties of hash functions that make it ideal for this project. They are: deterministic in nature, able to be quickly computed, pre-image resistance, small change in input changes the output, being a one way computation and lastly being collision resistant [20]. The first property of being deterministic refers to the concept that regardless of the number of times a certain input is put through a hash function and hashed, it yields the same output, every time. The ability to be deterministic is required because a student's records will be hashed and compared to check the authenticity of the records; if the hash of the student's record was to change every time it was hashed, the resulting output would have little to no meaning. Secondly, the hash must be relatively be quick in its computation, if it takes too long, it might not fit the execution speed and time requirements of an application. However, for this project, I

relied on two different hashing algorithms, the first was SHA-512 which was used for its ability to be quickly computed and the second being Bcrypt which can be modified to make it slower which can be helpful against brute-force attacks [21].

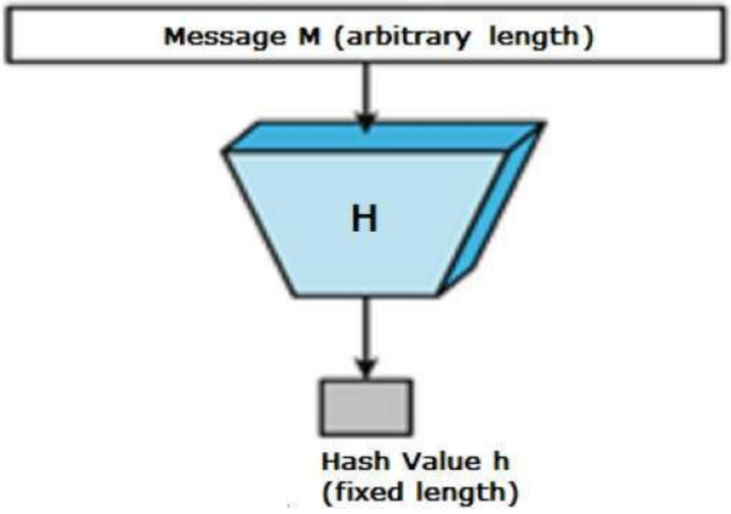

Figure 7 Hash Function Result Length

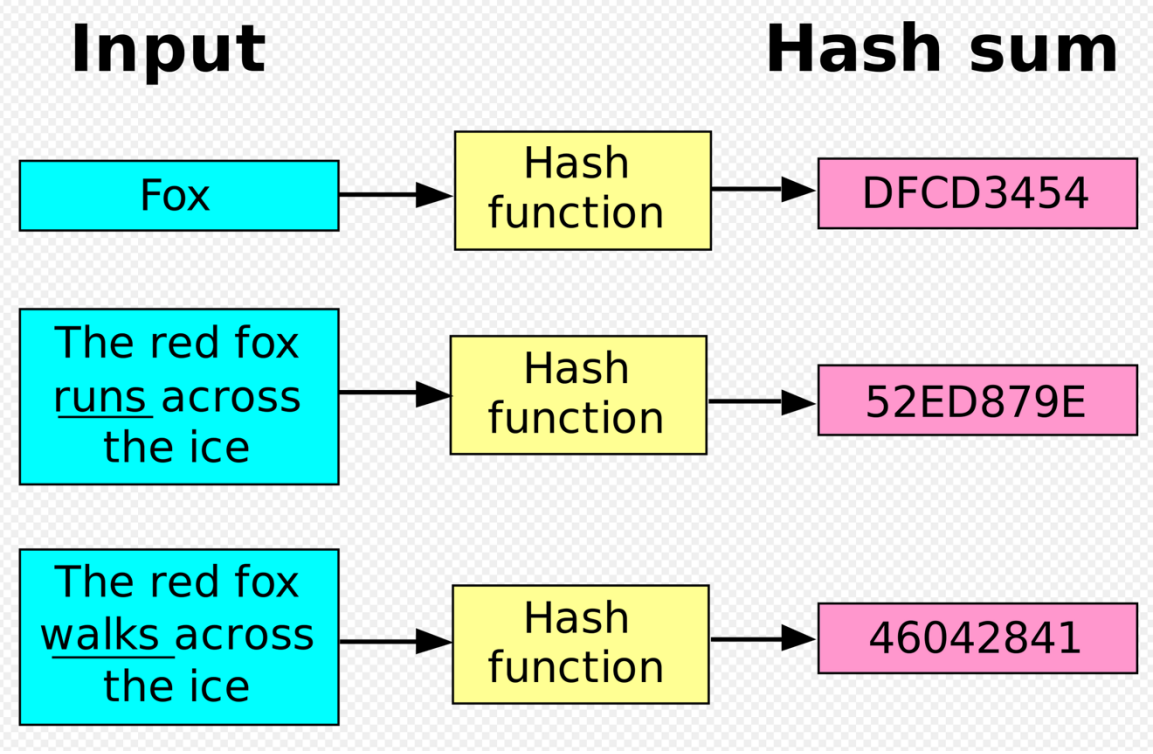

Figure 8 Hash Output

The concept of pre-image resistance states that given a function and its input, it should be unfeasible and impractical to determine the input. It is difficult to use concrete information such as impossible instead of words like impractical which allow for a bit of leeway because there have been hashing functions which have been broken and are no longer effective. If the input to the hash function is modified even by a bit, it is no longer uniquely equivalent to its old version and thus should yield a new output. It's a notion that just naturally lines up with the requirements of the hash function. A string is modified so that it is a new string, and the new string gets its own unique hash output. A hashing algorithm is a one-way mathematical function, you take an input and map it to an output; for the algorithm to be useful in cryptographical situations, it must be a concrete rule that it cannot be reversed. It the output was able to be reversed, it would defeat the purpose of it being used as a way to quickly store private information. Lastly, the property that is of utmost importance states that it should be infeasible for two different inputs to map to the same hash value. So, the hash of input A cannot ever be equal to hash of input B, each must have its own unique hash. If there is ever a case where they are equal, the case is called a hash collision. The existence of a case of a collision can lead attackers to exploit the weakness and perform malicious actions. The collision can be exploited by any application which compares two hashes [22]. For example, if a website checks the login of a user by comparing the hash of the provided password, it could be made possible that another carefully constructed password which yields the same hash be substituted in and passed to the server as an authentic login. The server would have no choice but to accept the provided information and server the requested information. Even in the most unsecure hashing algorithms, they were broken because

cases were found where collisions occurred, and those odds of those occurrences were extremely low, but still non zero. For a hashing algorithm to be taken as cryptographically secure it must not have any occurrences of collisions. As of right now, there are cryptographic hashing algorithms which are considered secure such as SHA-512 and Bcrypt, which have not been broken yet.

Hashing collisions are the primary way hashing algorithms are deemed broken. Hashing an input is not used to encrypt an input, it is inefficient and ineffective since a hash is a one-way function. There is no way to take the resulted output and put it back through the hashing algorithm to get the original text back. If one intends to do that, there are actual cryptographic functions which provide the desired encryption of plaintext and decryption of ciphertext.

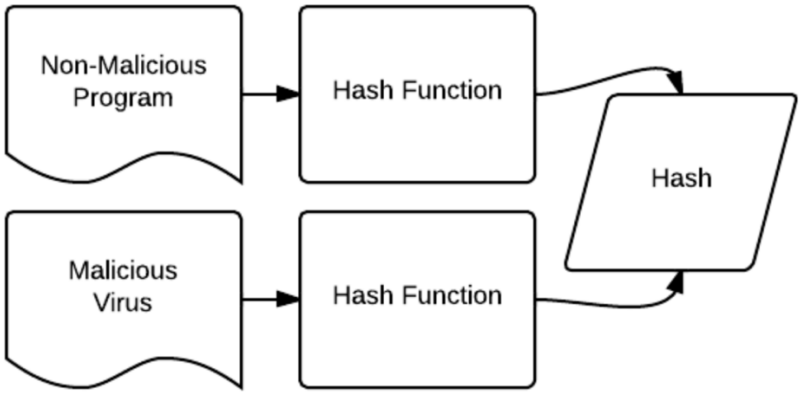

Figure 9 Hash Collision

A popular attack for a hashing algorithm is a rainbow table. A rainbow table is used for cracking passwords, or other sensitive data by precomputing the hashes for a predefined set of inputs [23]. For example, a rainbow table for the popular MD5 hash would look like a two-column table, where the first column has the input string such as the letters of the alphabet and dictionary words and the second column would have the

corresponding hash of that given input. Once you have a large enough rainbow table, it is fairly easy to search through the table to see if any of the leaked hashes in your possession are part of the table. In doing you, it is immediately apparent which string input yielded the given hash, thus giving away the original password or sensitive data. Such a tactic only works if the match occurs on the whole string, if a string you are in search of is a composite, then the rainbow table will be quite useless unless the computed hash in the table was generated from the composite string. Composing strings would have to be left up to pure chance because there are seemingly infinite combinations of strings, and the table would never be complete. In an essence, a rainbow table is a massive list of common passwords, or inputs and their hashes. A rainbow table is a step up from trying every possible combination of characters against the hashed password, the time taken to guess the correct choice increases exponentially as password length and keyspace increases.

User	Password
Stephen	auhsoJ
Lisa	hsifdrowS
James	1010NO1Z
Harry	sinocarD tupaC
Sarah	auhsoJ

User	Password Hash
Stephen	39e717cd3f5c4be78d97090c69f4e655
Lisa	f567c40623df407ba980bfad6dff5982
James	711f1f88006a48859616c3a5cbcc0377
Harry	fb74376102a049b9a7c5529784763c53
Sarah	39e717cd3f5c4be78d97090c69f4e655

User	Random Salt	Password Hash
Stephen	06917d7ed65c466fa180a6fb62313ab9	b65578786e544b6da70c3a9856cdb750
Lisa	51f2e43105164729bb46e7f20091adf8	2964e639aa7d457c8ec0358756cbffd9
James	fea659115b7541479c1f956a59f7ad2f	dd9e4cd20f134dda87f6ac771c48616f
Harry	30ebf72072134f1bb40faa8949db6e85	204767673a8d4fa9a7542ebc3eceb3a2
Sarah	711f51082ea84d949f6e3efecf29f270	e3afb27d59a34782b6b4baa0c37e2958

Figure 10 Hash Rainbow Table

One way to mitigate a rainbow table attack is by adding a salt to the hash. Hashing by itself is simply not enough to protect passwords or other fields. Salts are short random characters that are appended to the end of the password before they are hashed. So, a combination of the password and salt will thwart any rainbow table attack as long as the salt is also not leaked. In general, the salt is stored as plaintext alongside the hash of the password, so that the authentication system knows what the salt was and can verify the hash of the password plus the salt. It may seem counter intuitive that the salt is stored in plain view since you are essentially giving the hacker part of the password, and dictionary attacks and brute force can still be a problem if the hacker knows where to put the salt in the guesses However, rainbow attacks will be ineffective due to their nature. Additionally, most developers also make use of a pepper. A pepper is a short string or character appended to the end of a password. By their nature, peppers are random and different for each password and it is common for them to be around a character or two. So, the new hash stored is the hash of the password, appended with a random pepper and the salt. Suppose this system is used for a login system, a user will enter their username and their password, and the system will append all the possible combinations of the pepper and the designated salt and generate the hashes of every combination until one matches the stored hash. If the hashes match, then the user is allowed to log into the system. The whole point of this is that the pepper is not stored. No one knows what the pepper is, not even the website, until it goes through all the possible options to find it. All of this leads to extra time taken when compared to a system using no pepper. The use of a pepper is not an issue to the user, since they do not know the underlying strategy for

hashing, however, it adds an extra layer of complication for anyone looking to crack the system.

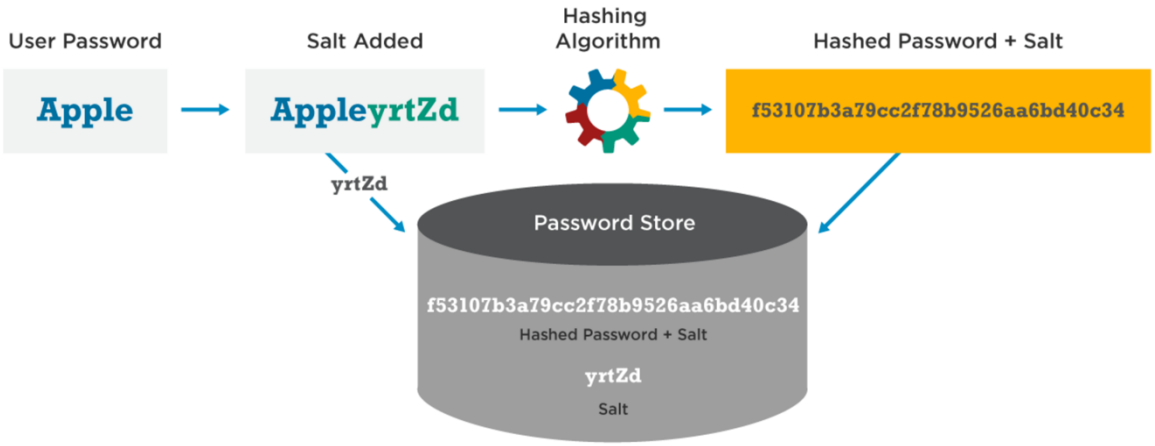

Figure 11 Password + Salt

The takeaway idea from hashing is to avoid any novel homemade hashing functions or methods that people invent and to avoid reused values. Many developers insist on designing their own hashing functions and think that the concept of secure cryptographic design means to throw together every kind of cryptographic and non-cryptographic operation that can be thought of [24]. The developer thinks that because their own function is over engineered and complex, that the attackers themselves will be unable to crack their methodology. In reality, it's only the developer who might have confused himself due to the complexity rather than the attacker. Anyone wanting to make use of a password hashing function should look into the currently standardized hashing functions such as SHA or Bcrypt. At one point in time, MD5 was a popular hash function, however, it is now considered broken. It was commonly used for password hashing due to it being very fast. The primary goal of the salt is to be as unique as possible; the usefulness of salt goes down if the salt value is reused anywhere.

Chapter 5: Technologies Used

Throughout this project, there were various technologies that were utilized for the successful execution of the workflow of the project.

The front-end website app makes use of HTML, JavaScript with the CSUN website template to match the look and feel. The application is intended to be hosted on Microsoft Azure App Service, if the entire Microsoft stack is used. The frontend is a Node.js app which routes the form data to a simple Azure Function URL, and receives the pdf binary blob and reconstructs it to be displayed to the user. The simplicity and portable nature of the app allows for the hosting to be on any service which allows hosting of Node.js apps, thus bypassing the typical vendor lock-in.

The main part of the project was the Azure Functions Service. The Azure Functions were implemented in Microsoft's C# programming language. The code responsible for generating the pdf that would be returned to the user, utilized the DinkToPdf wrapper library [25]. DinkToPdf was immensely useful since it allows the user of the library to design the layout of the pdf using html and model out the layout of the file. Additionally, Microsoft's Azure Documents library was used to query Cosmos DB and execute other requests against the service. There were two hashing algorithms that were employed in the project as well.

Initially, the project was designed with a single hashing algorithm known as the Secure Hashing Algorithm-2 (SHA-2). More specifically, the SHA-2 family consists of functions with various length hash outputs, the specific length used was SHA-512. It is one of the industry standards used when the primary concerns are security and speed [26]. The second hashing algorithm was chosen with security and slowness as the

primary attributes. The algorithm that best fit the criteria is known as Bcrypt [21]. It is the default hashing algorithm in some Linux distributions and on OpenBSD operating system. It is resistant to brute-force search attacks because one can specify the number of iterations if must do in its calculations, thus making the algorithm slower.

The administrator's utility tool was created in C# and the Windows Forms graphical class library, which is part of the Microsoft's .NET Framework. Microsoft's graphical framework allowed for the quick prototyping and testing of the parts needed in this project. The benchmarking was performed with BenchmarkDotNet [27]. BenchmarkDotNet is an easy to use benchmarking library that generates reports in various file formats and allows for multiple iterations of the code in order to measure the performance once the code is warmed up. Lastly, the JSON data storage format was chosen to represent the class models, and the Newtonsoft's Json.NET framework was used extensively due to its serializing and de-serializing capabilities [28].

The testing of the project was carried out locally on machine using Visual Studio 2017 on Windows 10 operating system. Once the project's dependencies were chosen and installed, their versions were frozen for maximum future compatibility and stability. The project was tested progressively throughout its many phases making use of Microsoft's Cosmos DB Emulator, Azure Functions offline deployment and the API testing software Postman [29]. Testing was done locally to avoid the costs of utilizing Azure services for a proof of concept project.

Chapter 6: Project Implementation and Workflow

The workflow of the project will be described from the perspective of the end user and the perspective of the administrator. The overall high-level workflow leads the data from the front-end website app to the function responsible for searching the database for the student's academic information and compiling said information into a pdf and transmitting it back to the user so their requested pdf file can be displayed in the matter of a few seconds. The alternative workflow showcases how the student's data is organized, compiled, uploaded to the database, and secured against any future tampering. Each section will also touch upon the idea of accomplishing this in a secure manner on a serverless architecture.

6.1 End User Workflow

As stated in the introduction of this section, the workflow for the end-user, be it the student themselves or a business requesting the student's records, was designed to be simple and straightforward.

The user starts off by navigating to the front-end website. The only information required from the user is the student's first name, middle name, and last name. Those three key pieces of information were chosen in order to differentiate students who might have same first and last names. A fourth field for the date of birth was added and tested, but was later removed from the options due to not needing in, and to speed up the testing process.

Express is a Node.js minimalist web framework that provides capabilities to handle different requests and act as an URL router. It allows the user to quickly develop

API's and web apps. Express was used to handle any erroneous, and unauthorized paths; if the user navigates to any part of the domain that does not include the index.html file, they are sent back the file which holds the HTTP 404 Page Not Found error message. When the Submit button is pressed on the form, the information from the three form fields are passed to the Express framework to be handled, the language of usage here is JavaScript. An object is created that holds the first name, middle name, and the last name and a JSON string is created representing the object. Another object is created to hold the HTTP POST request details. The details include the URL of the Azure Function which receives the request and searches the database for the student record. The request is then submitted to the Azure Function. The web app is secured via the HTTPS encrypted connection. There is security built in to the system when the web app is hosted on Azure App Service, because Azure automatically applies an SSL certificate and an Azure Function can be hosted with HTTPS enabled as well. Thus, enabling encryption on both sides, allows the student's name information to be transmitted securely in the HTTPS request.

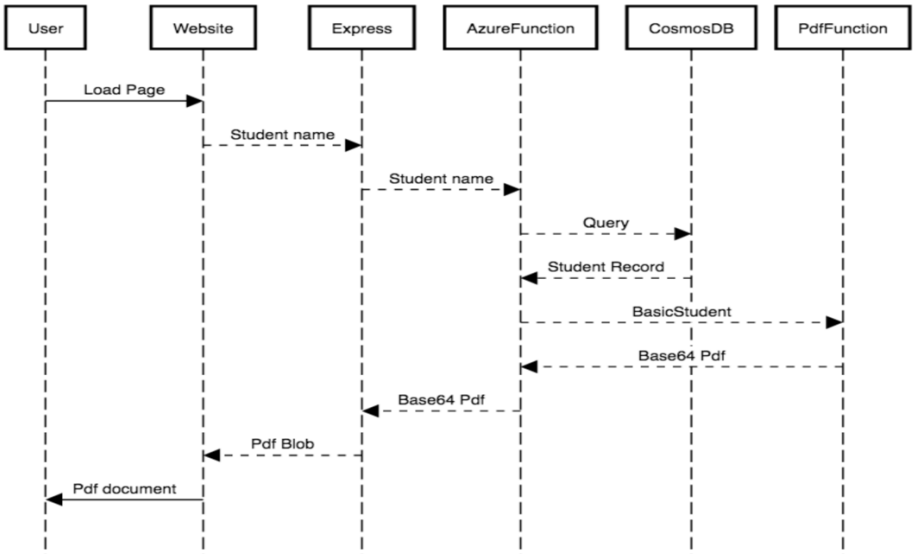

Figure 12 End User Workflow

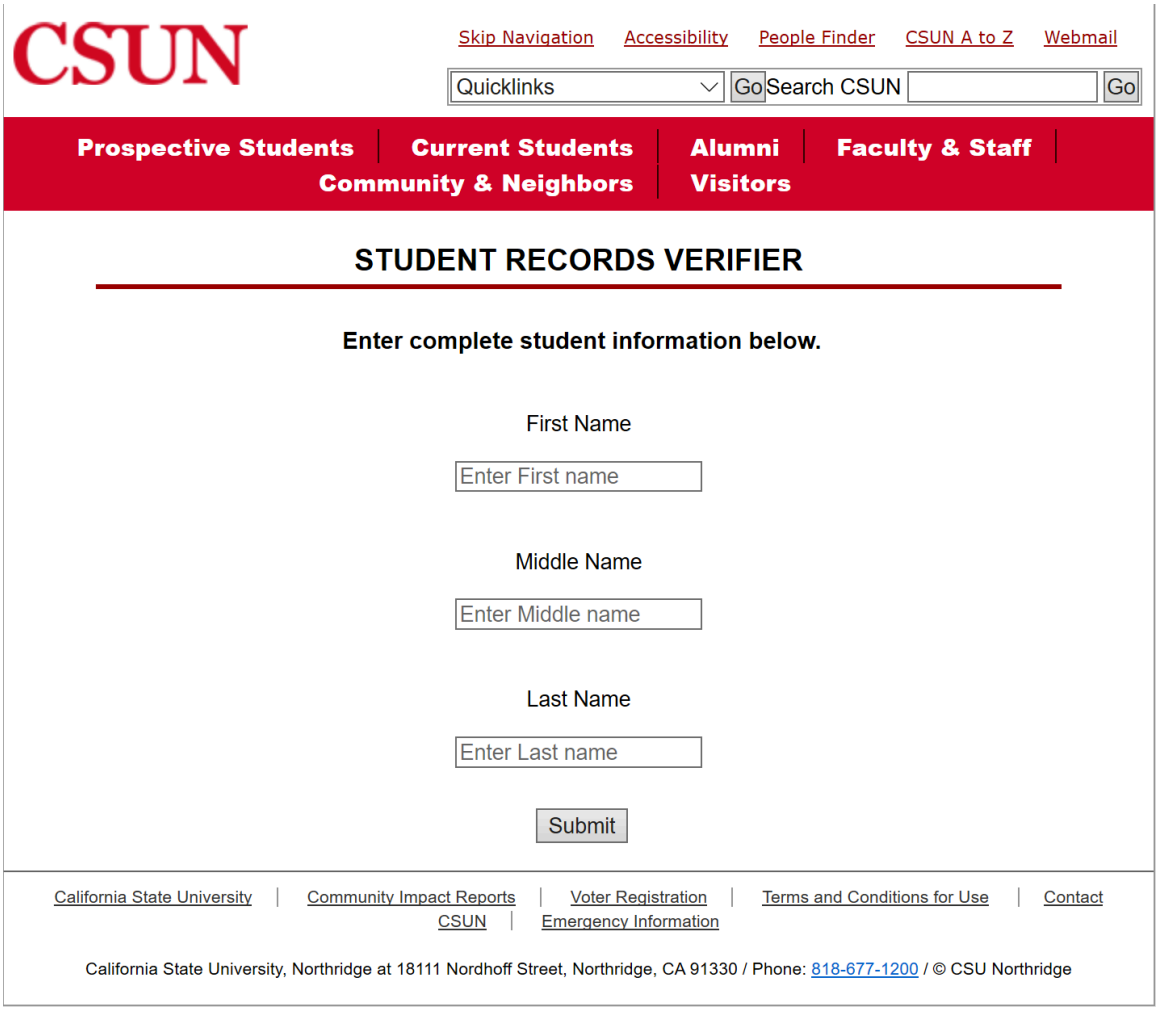

Figure 13 End User Website

Much of the heavy lifting is performed once the Azure Function receives the request with the student's information. The request JSON data is parsed for the student's first name, middle name and last name and deserialized in an object class called Student. The Student class holds just three string fields to hold the parts of the name. Additionally, each name's first letter is capitalized so that it matches the format of the name as stored in the Cosmos database. Next up, an instance of the database connector class called CosmosConnector is created. The CosmosConncetor implements an interface called IDatabaseConnector which contains fields for the URI, access key, database name, table name and a Microsoft DocumentClient object, which is responsible for the methods that

allows interaction with the Cosmos database. The interface also defines some methods that must be implemented by any class that implements the interface. An interface is like a contract between any class implementing it and itself. Interfaces are used to group objects and a way to allow a class to inherit from more than one class. Since an interface acts like a contract, any class implementing the interface must also include and provide direction for any fields or methods defined in the interface. The benefit of using interface include loose coupling and allows for plug and play component functionality. The CosmosConnector class implements the IDatabaseConnector and has to implement functions such as GetStudentRecords, CreateTable, and CreateDatabase. However, there can be another class called GoogleDatabaseConnector or AmazonDatabaseConnector which will have to override methods of the interface, yet their implementation will be vendor specific. By implementing interfaces in this project, it adds an ability to be portable if the cloud vendor is changed. The business logic will still perform and act in the same manner, the only difference will be that the cloud specific logic would be different. The database connector makes a connection to the Cosmos database, and searches the StudentRecords table for the student with the same name. The method responsible for the search has a single string which contains the SQL query. In general, the SQL query should not be built out of strings since it allows for SQL injection attacks where someone looking to exploit can inject additional commands. This issue was mitigated by replacing the variables values in the expected query command. The returned data is a dynamic datatype which will contain the entire student record as stored in the database. The data includes additional information that the database inserts such as an internal record number, and last modified date value. In C#, the dynamic type allows the

compiler to skip compile time checking and determine the type at run time. It was beneficial in packing the information into the dynamic type and then extracting information into a proper model type. The received dynamic information is first stored dictionary data type so that the student information can be accessed in the dictionary and be mapped to an object of the BasicStudent type. Throughout this project, there are two essential classes: BasicStudent, and FullStudent. The former is used to represent the basic information of the student that will be returned back to the end user. That information includes: first name, middle name, last name, date of birth, school organization, school division, name of degree, name of major, and date of the awarded degree. Once the BasicStudent object has been constructed, it is supplied to an instance of the Pdf class via constructor-based dependency injection. Dependency injection is a technique where the required object is passed in as a parameter to the class that requires it. They can be passed in by various means, this project makes use of constructor injection. The Pdf class is responsible for generating the model of the file that will be later returned. It takes in the BasicStudent object and the accompanying information and injects it into a string that is included in a simple html template that will be rendered into the final file. The pdf file is then generated in memory and saved to a byte array; the byte is then converted to a string that is encoded as a Base64 string. Base64 is an encoding scheme that represents some binary data as a string consisting of ASCII characters. ASCII is an older standard that was used to represent characters in the English language. The encoded Base64 string is returned back as a response if the student was found; otherwise, a HTTP 404 response is returned if the student did not exist within the database.

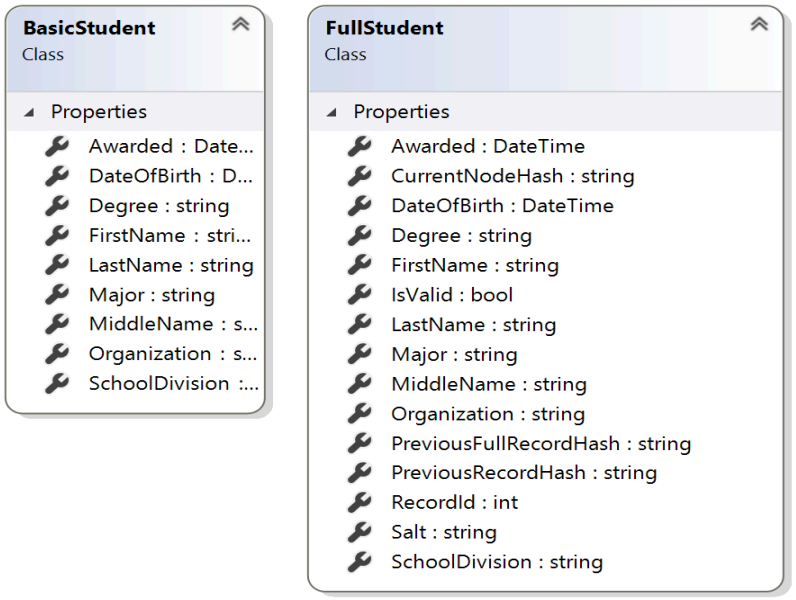

Figure 14 BasicStudent and FullStudent

Once the response containing either the Base64 encoded pdf file or the not found status code is received by the web app, it is extracted and stored in an instance of Node.js Buffer class. The Buffer class allows the developer to specify the data, its encoding and allows of interaction with streams of binary data. Essentially, it allows for recreation of the pdf file by specifying its encoding. The web app readies a new response back to the end user, it specifies the content type to be of type application/pdf and the response is sent back to the browser. Once the browser receives the response, it automatically interprets the content type and the stream containing the Base64 pdf file and displays it as a fully rendered pdf file. The rendered pdf file displays the information in a manner similar to how the current system handles it. The entire process of entering the student's name and submitting the information, constructing the file and the display of the file takes a few seconds on average.

California State University, Northridge

DegreeVerify Certificate

Date Requested:

3/31/2019 11:29:59 PM

Information Verified

Name on School's Records:

Karandeep Singh Bhamra

Date Awarded:

1/28/2019

Degree Title:

Bachelor of Science

Name of School:

California State University - Northridge

School Division:

College of Eng/ Comp Sci

Major Course(s) of Study:

Computer Science

Figure 15 Project Verified Certificate

National Student Clearinghouse®
2300 Dulles Station Blvd., Suite 220, Herndon, VA 20171
PH (703) 742-4200 FX (703) 318-4058
www.studentclearinghouse.org

DegreeVerify Certificate

Transaction ID#:	0178666596	**Date Requested:**	01/13/2019 04:05 EST
Requested by:	KARAN BHAMRA	**Date Notified:**	01/13/2019 04:06 EST
Status:	Confirmed		
Fee:	$24.95		

INFORMATION YOU PROVIDED

Subject Name:	**KARANDEEP** First Name	**SINGH** Middle Name	**BHAMRA** Last Name
Name Used While Attending School: (if different from above)	**KARAN** First Name	Middle Name	**BHAMRA** Last Name
Date of Birth:	**06/25/1991** mm/dd/yyyy		

School Name: **CALIFORNIA STATE UNIVERSITY - NORTHRIDGE**

Degree Award Year:

Attempt To: **Verify a degree**

INFORMATION VERIFIED

Name On School's Records:	**KARANDEEP SINGH BHAMRA**
Date Awarded:	**05/26/2017**
Degree Title:	**BACHELOR OF SCIENCE**
Official Name of School:	**CALIFORNIA STATE UNIVERSITY - NORTHRIDGE**
School Division:	**COLLEGE OF ENG/COMP SCI**
Major Course(s) of Study:	**COMPUTER SCIENCE**
(and NCES CIP Code, if available):	**110701**

Figure 16 Original Verified Certificate

6.2 Administrator Workflow

The workflow of the administrator that is going to be employing the utility tool and interacting with the database is quite different from the end user's workflow. To begin, there is no web app, instead, it is a single application with a graphical user interface with six buttons which launch separate forms for different actions. The application is called the Student Record Tool, and for rapid prototyping purposes was created using Windows Forms framework and C# language. The process starts with creating a user record, and then leads to adding the record to the database and provides the user of the tool the ability to verify the integrity of the student records.

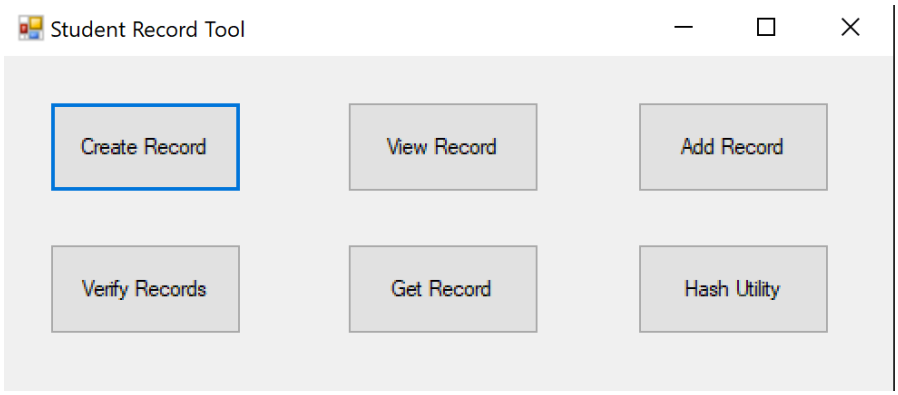

Figure 17 Student Record Tool

Initially, the user of the tool would want to create an individual student record. When the corresponding button is clicked, it launches a new form that contains the textboxes that will contain the information that goes into a BasicStudent object. Once the fields are filled out and the generate record button is pressed, there is a quick validation process that is carried out to make sure the fields are not empty. A temporary BasicStudent object is created which is then serialized into a JSON string. The JSON string is saved into a json extension text file with the filename being the full name of the student. This functionality was created to quickly create a student record and test the

adding of the record to the database. Ideally, on the production version, the student record would be stored in a format that would allow the administrator to automate the creation of records by reading the records. The National Student Clearinghouse is the firm that is responsible for the current CSUN degree and attendance verification; they were contacted in order to obtain further technical information on how the student data is transmitted and to gain knowledge on their inner workings so that this project would be able to implement a system that would emulate theirs. Unfortunately, their response stated that they would not be able to disclose any operating or technical information, thus, this project makes educated presumptions to solve concerns at certain parts of the process.

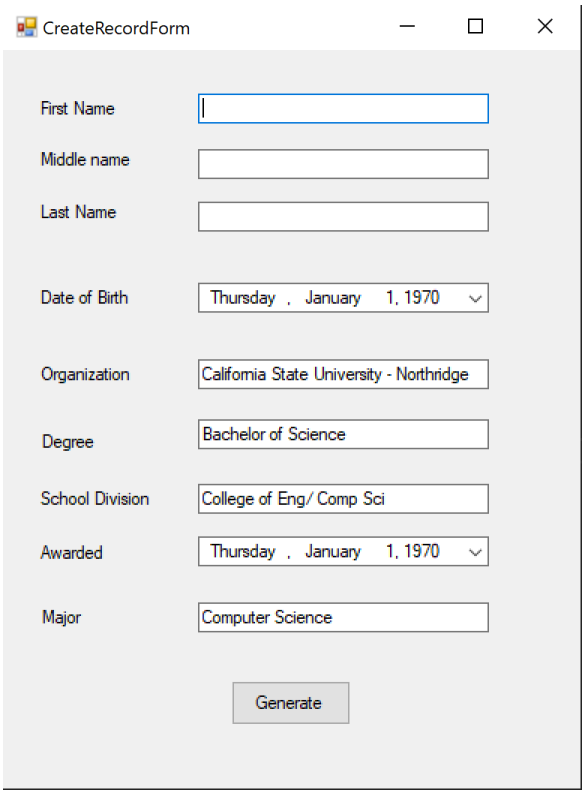

Figure 18 Create Record Form

The second button is responsible for viewing the file containing the generated student record. It launches a file browser that can open a student's generated file and

displays it on a form identical to the one displayed in the create record form, except all the fields disallow editing and are in a read only mode.

The third form launches a form that behaves in the same manner as the previous button except that it adds an additional button on the form that will insert the student record into the database. When the add button is clicked, the student file is loaded converted into a BasicStudent object, which is then serialized into a JSON string. The JSON string is added to an HttpWebRequest object which contains the URL of the second Azure Function and is then transmitted. The second Azure Function is responsible for receiving a student record and adding it to the database. Once the Azure Function receives the request, it parses the request body and deserializes the student information into a BasicStudent object. There is a CosmosConnector object that is created, that will be utilized to access the database. The connector runs a function that will create a database called StudentDatabase if it does not exist. Secondly, it will create a table called StudentRecords if it was previously not created. A request is made to the database to retrieve the last inserted record. If the last record yielded a non-existent value, then a genesis student record is inserted. In blockchains, a genesis block or node is the very first block in the blockchain [30]. The genesis node is generally hardcoded because it does not reference a previous node. For this project, the genesis block is a student record for the school mascot Matty the Matador. The next step involves expanding the BasicStudent object into a FullStudent object. FullStudent contains all the fields from the BasicStudent and adds additional fields such as: record id number, hash of previous record, current node's hash, salt value, and the hash of the previous record as a FullStudent object. There was a utility class called Hash which is responsible for generating hashes, and salt values.

The Hash class also has overloaded functions which can generate the hash and salt values using SHA-512 or Bcrypt hashing algorithms. To calculate the previous node's hash for the genesis student, a string with the text "test" was used. To calculate the hash of the current node, the FullStudent node is converted to a JSON string and the string's hash is calculated. In earlier testing, the FullStudent was converted to a byte array and hash was taken of that byte array; however, it was discovered that there were some instances where the byte array different on subsequent runs due to the way some systems handled memory allocation. The genesis student's final hash value is calculated by concatenating the initial hash with the hash of the random salt value. A utility class called StudentMapper provides the ability to map BasicStudent into FullStudent and vice versa. StudentMapper was used to get the FullStudent object representing the genesis block. The FullStudent object representing the genesis block is then passed into the connector function responsible for inserting the record into the database. Once the genesis block is inserted, the current student full object is constructed and inserted in the same manner as the genesis block. If the student was successfully inserted, a success method is returned to the add student form in the utility tool; otherwise, a failure response is returned.

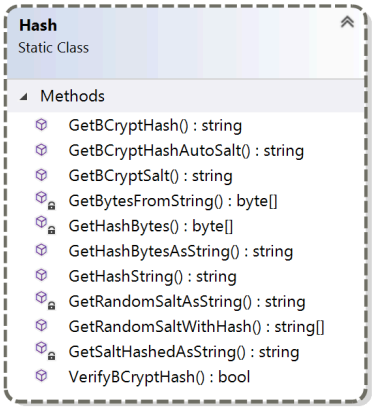

Figure 19 Hash Class

In the utility tool, there are two service forms that were created in order to aid the administrator. The first form takes the full name of the student and returns the FullStudent object representing the student record. The record is then also mapped to a BasicStudent object and the JSON representation of both the FullStudent and the BasicStudent objects are displayed on separate multiline textboxes. The intention of this form was the let the administrator view the data representing the given student similar to the way the data is represented and shown in the Cosmos database explorer. The second of these utility forms is one that takes in some input string and will generate the inputs hash string based on the options of SHA-512 or Bcrypt and various different worker values. When both forms are used in tandem with each other, they allow the user to debug the chain of student records in a quick and efficient manner.

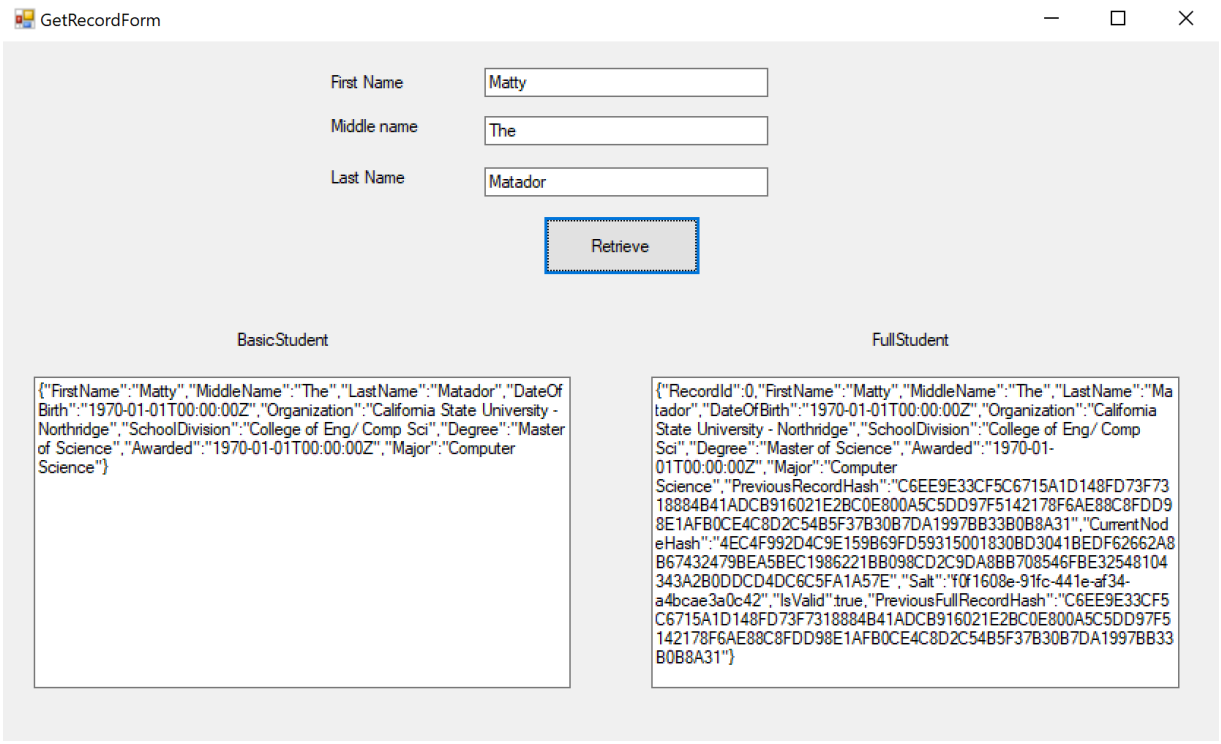

Figure 20 Get Record Form

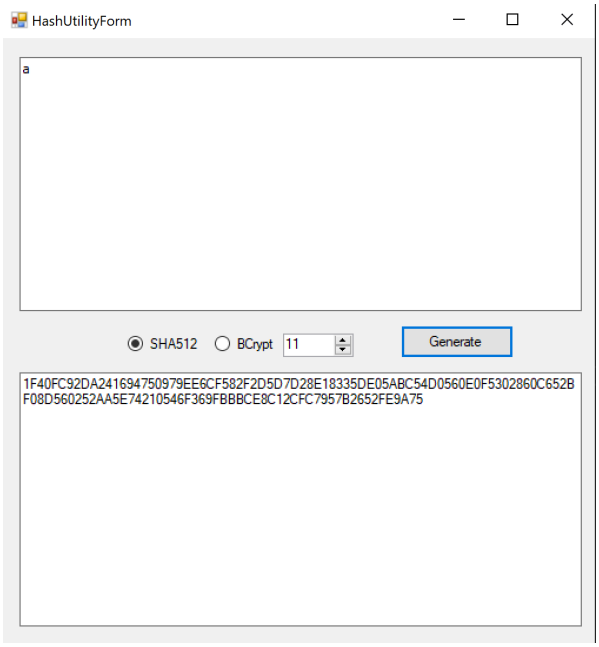

Figure 21 Hash Utility Form

The last and the most important functionality of the Student Record Tool involves its ability to verify and validate the chain of student records. It is impossible to know if a student record in the database was modified if we look at just the record. There is no ability to compare it with a previous version to see if the record was altered. It is easy to compare if there are two versions of the record, it's a simple comparison to see if there is a mismatch in any of their fields. However, storing two versions of a single student record is inefficient, it will utilize double the storage space in the database and still falls victim to the same problem. The idea of verification and validation of the student records comes from cryptocurrency blockchain. In a blockchain, each node stores a hash of the previous node, thus, only a comparison of the stored hash against the recalculated hash is needed. If the hashes do not match, it is easy to know that the previous was modified from the original record. In the utility tool, the last button launches the form containing the validation and verification logic. There is a button that commences the validation process, a label that displays the number of students in the database and a progress bar

that showcases the progress of the verification process. The process starts by connecting to the database and adding the FullStudent record for each student into a list. The logic for the verification process dictates that each student record must be iterated over and the newly computed hash of the previous record checked against the hash stored in the current record. There was always a small possibility that the student record's might be altered by an individual with suspicious motives and they would compute the new hash for the newly modified record and update the field in the next record. To overcome this small, but severe flaw, the records store two separate hashes. The verification loop starts at the end of the list and recreates two objects, the FullStudent and BasicStudent for the current node and the previous node. It then computes the hashes for the two objects and compares them against the stored hashes. It is much more difficult to exploit a double verification system because the next record stores two separate hashes of the current node, and every subsequent node's hash is dependent on the hash of the previous node. So, even if the hash of one of the nodes was to be chained, the logic responsible for checking the hash matches would detect the mismatch and halt the process on the given student record. It is still impossible to claim to be fully secured, there is always a small chance that any security system can be foiled. To overcome that issue, the hashing function can be made to be so computationally expensive that the effort to modify and update the remainder of the chain of records is not a trivial manner. By employing the Bcrypt hashing function, it can mitigate sophisticated attacks by increasing the iteration count of the hashing function so that it is very computationally intensive. Therefore, it can discourage anyone seeking to calculate the new hashes for the modified student and updating the linked hashes in the remainder of the records. Once all the records have been

verified, the entire chain of records can be considered validated, otherwise, the verification process halts at the failing student where the mismatch occurs.

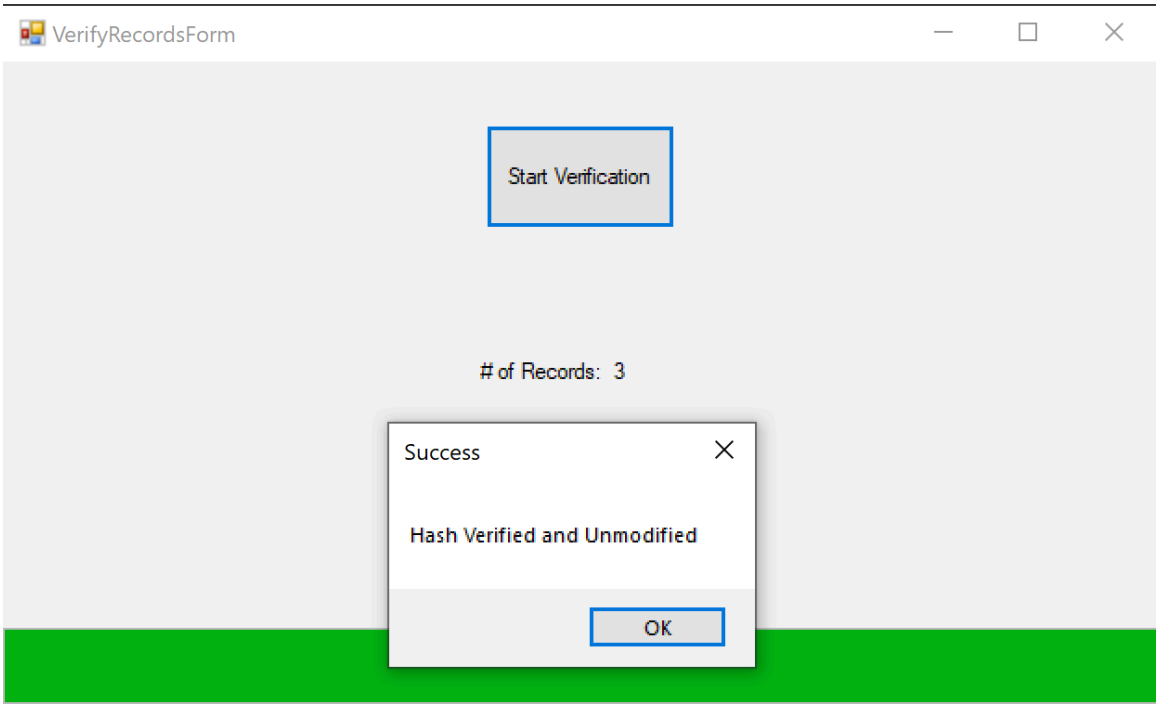

Figure 22 Verification Form

Hence, we took an in-depth tour of the Student Record Tool and listed how its various functions would allow the administrator to interact with the database and the records. For the purposes of this project, it is limited to running on Microsoft Windows operating system due to the Windows Forms framework; the functionality can be adapted to allow for an implementation as a web-based application so that any operating system can be used and the principles of serverless web architecture can apply to it as well.

Chapter 7: Project Results

Overall, the project accomplished every reasonable requirement that was set forth, and it managed to meet all the goals. In this case, the goal was to come up with a solution in which the academic institution would create and operate their own student record verification system. The system would borrow heavily from modern technologies such as blockchains, for the purposes of this project, the concept of a blockchain was utilized. Additionally, there would need to be a system which would allow for the verification and validation of the student records to maintain the authenticity of the academic records. The primary goal of this project was to cut out the middleman firm which is currently responsible for the verification of academic records at the university. There is no need for the data of the student to have to go through a third-party company; the data should only be exchanged between the university and anyone looking to verify the academic records. In order to popularize the idea that the academic institution should operate such as system, there was a great need of some incentives. Effective decision making requires comparing additional costs of a replacement system with the additional benefits. Since cost is the biggest incentive, it was decided that a serverless architecture would be adapted for this project. A serverless architecture allows the platform to do more all at a reduced cost. Going serverless is a great way of cutting costs because it allows for the outsourcing of responsibilities related to the self-managing of the infrastructure. Additionally, in serverless computing, the costs are determined by actively used resources. If the application is used few times a day, then the cost is computed for the duration of those invocations. Serverless architecture also solves the problem of scalability for free. Scaling is an automatic feature and the infrastructure expands

automatically to utilize more resources if needed to handle the additional growth. Lastly, a self-operated service allows the institution to take back control of its student's records. There is no privacy in the data if it is handed over to some third-party firm. It should only involve the academic institution, the student and anyone seeking to obtain the student's records for verification.

There were several areas of notable success in this completed project. The first area was the simplified web application that the user would be comfortable using. The familiarity is brought upon by the design matching the school's current design. Furthermore, the mandatory information requested in the web application is the same as the current verification service. The goal of the project was to emulate the familiarity and improve upon the current system.

The second area of success was the resulting certificate generated by the system and displayed back to the user of the web application. The certificate generated contains the same basic information as the one provided by the current system. The biggest differences are that the current system charges a fee of roughly twenty-five dollars, and the one created by the project does not have a fee associated with it, and does not pass on the costs to the student. The current system charges the academic institution to operate and provide the service of record verification and they charge students or anyone looking to verify the records. The proposed alternative cuts both of those costs, there is no proposed fee to anyone seeking to verify the records and the operating fees are significantly less due to the serverless nature of the service.

The final success was the implementation of the blockchain like data structure to hold the student records. Initially, the task seemed insurmountably large; however, once

the problem was broken down to multiple smaller tasks, they became much more manageable. Once the basic student information was inserted into the database, the task of hashing and storing of nodes became the next problem. It too was broken down into a smaller problem where it would involve the generation of the hash of the node. Next up, the hash would be linked into the next inserted record. The double hash verification system was a non-trivial matter to solve. Once all the parts of the validation system were connected, it worked flawlessly.

The toughest part of this project as relaxing expectations. Initially, the goal was to create an actual decentralized blockchain to store the student records. This quickly proved to be too difficult for a single person to pull off and expectations were scaled back. It took another approach at the blockchain problem to realize that the project would be very much a centralized, as it would be run by the academic institution and it would be hard to adapt it to decentralized platform. This problem led to the solution of treating the database in a manner similar to a blockchain where each record is linked to the previous node with a cryptographically secure hash.

No project is also without false starts, and this was no exception. Initially, this project was based on Amazon's AWS platform due to familiarity with their offered services. However, the lack of offline software development kits and testing tools led to a complete restart with Microsoft's Azure platform. There was some time wasted spent relearning the counterpart services and tools, however it in the end it became one of the biggest advantages due to how well Microsoft has designed their platform to integrate with their tools. Additionally, the documentation provided by Microsoft on their tools and

services is far superior even though it is a less mature platform compared to Amazon's AWS.

7.1 Comparison of Hashing Algorithms

The two algorithms that were used in this project were SHA-512 and Bcrypt. The SHA algorithm is considered a fast hashing algorithm. It is a cryptographic hash whereas Bcrypt is a password hashing function, also known as password based key derivation function (PBKDF). The SHA was designed to be fast by nature. Since Bcrypt is a password hash, it needs to perform key strengthening on the input. In cryptography, key strengthening is a technique that secures a weak key or password to be more resilient towards a brute-force attack [31]. It does so by increasing the resources it takes to test each key. The passwords used by humans are often short and based on dictionary words, which makes them easy to crack via simple brute force methods. Key strengthening is a deterministic function that will yield the same longer, enhanced key for the same weak key. Bcrypt performs key strengthening by slowing down the calculation so that the attacker has to spend more time to find the input in a dictionary attack. Bcrypt also employs a work factor, which is how many rounds of strengthening the input goes through. The work factor results in a number to the power of 2. The default work factor is 11 in the official C# library, so the minimum number of rounds Bcrypt can through is 2048. Additionally, Bcrypt also includes a salt as the input, SHA algorithm does not require that. So, Bcrypt is designed to be able to thwart rainbow table attacks whereas the SHA is vulnerable to them. SHA is an algorithm that also benefits from being implemented on the graphics processing unit. The amount of time it spends computing

hashes can be sped up considerable on the GPU as opposed to the CPU. If the attacker knows that the system is implemented with SHA in mind, they can be at an advantage. Bcrypt was designed to be slow and there is no increase if it is implemented on the GPU. SHA can be implemented with multiple rounds like Bcrypt; however, it was not intended to be a password hash. Developers should take the time and compare their needs and pick the right hashing function.

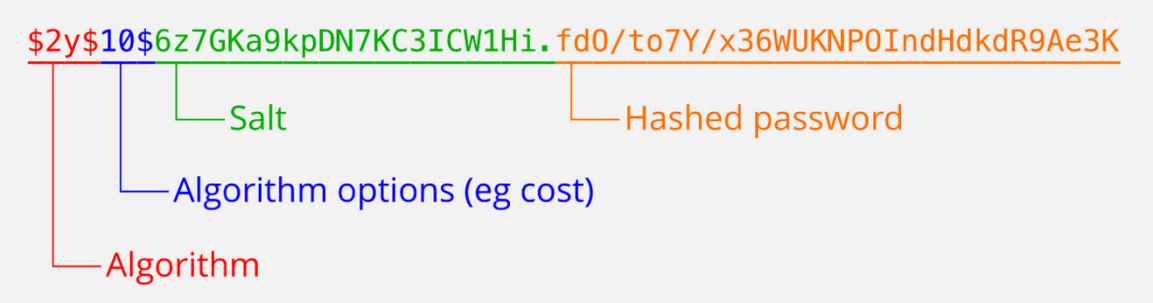

Figure 23 Bcrypt Value Breakdown

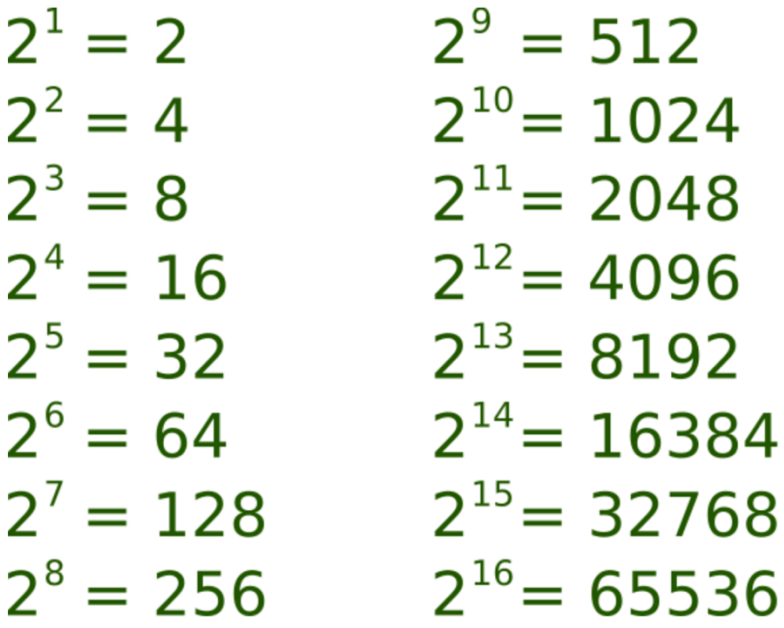

Figure 24 Bcrypt Worker Value Round Increase

7.2 Technical Limitations of Serverless Architecture

When employing Microsoft's Azure architecture, it is important to acknowledge its technical limitations. Azure Functions only bill you for the resources you actively consume, and they calculate the cost based on the execution time and the memory usage for each function call. Azure Functions do not provide its users with unlimited resources, there are some severely limiting factors that the user must be aware of. For starters, there is a default value of five minutes for the timeout duration, it is able to be overridden and set to a maximum value of ten minutes. So, if the code one is running on the Function does not complete in ten minutes, it is forcibly shut down. An HTTP triggered function cannot also take longer than four minutes because it is the default idle timeout of the Azure Load Balancer [32]. These values are what Microsoft determines to be a reasonable response time for a web request. There is also a maximum request size of 100 Megabytes and the length of the request URL can be a maximum of 8192 characters long. Additionally, the Function cannot allocate more than 1.5 Gigabytes of memory per instance, and the local temporary storage is limited to 1 Gigabyte [32]. There are similar service limits for Azure App Services which can host the project front end website. In comparison, there are various tiers for Web Apps and they range in storage limit from 1 Gigabyte to 1 Terabyte. The CPU time is can also be configured to range from a minimum of 3 minutes on the free plan to unlimited on the mid to high level tiers. The developers must keep these limitations in mind when comparing the services provided by various cloud vendors.

7.3 Benchmark Results

There were various aspects of the project that were tested. Testing was performed with the C# benchmarking library BenchmarkDotNet. The first test showcases the difference in time taken between the Bcrypt, SHA512 and the C# standard library hashing algorithms. The Bcrypt algorithm is further tested with various worker values ranging from 11 to 15, where each subsequent round greatly increases the time taken to generate the hash. From the data, SHA512 and the default hash algorithm are near instantaneous, whereas, the Bcrypt highlights its deliberate choice to be a slow hashing algorithm.

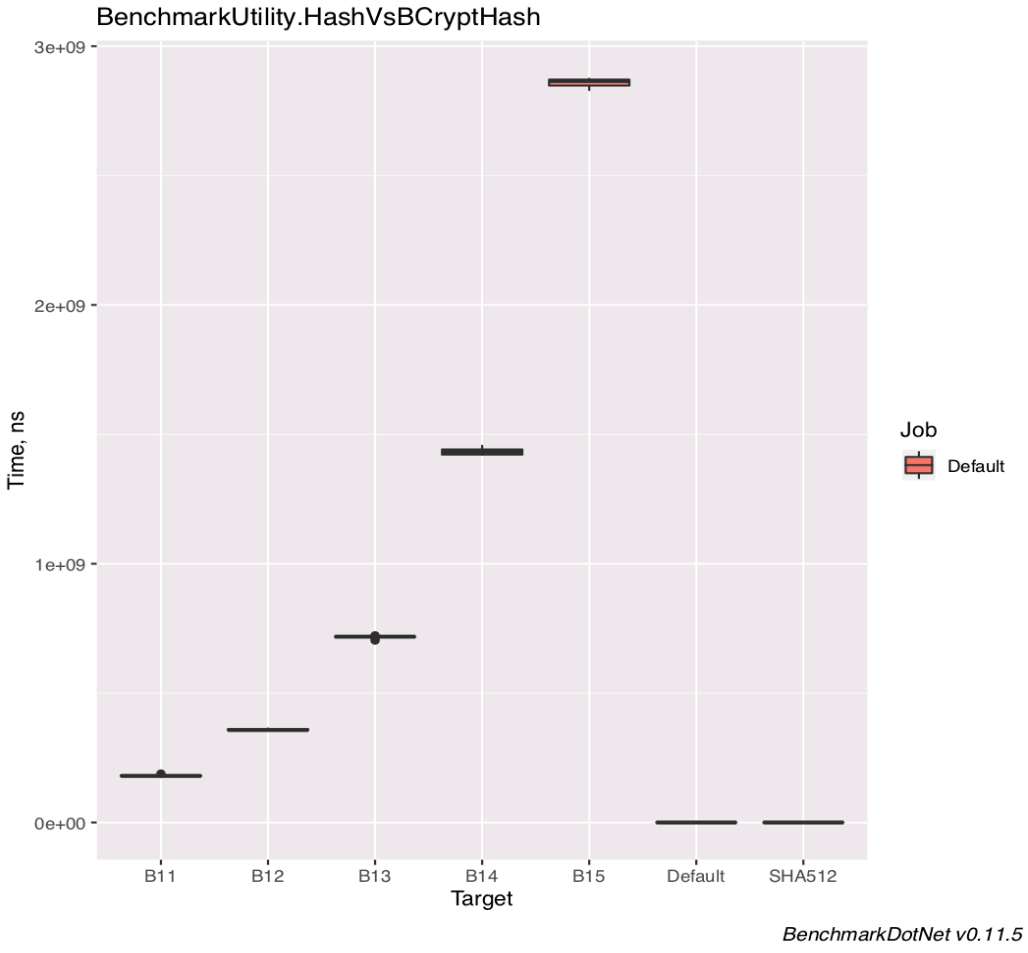

Figure 25 Bcrypt vs SHA512

Secondly, the amount of time it took the Azure Function to insert a student record into the Cosmos database was measured, along with the amount of time it took the Azure Function to generate and return the pdf file back to end user. Each test was run 8 times. The tests expose one of the biggest downsides of serverless function, which is the cold start time. As we can see, the initial run is fairly high for both functions, however, each run thereafter is cut down significantly. Cold start describes the idea that an application that hasn't been used in a while will take longer to start up. The cold start problem is a minor inconvenience for this project as it only adds a few additional seconds to the application run time, and isn't too big of a deal for most apps. This issue can be bypassed by setting up a pinging service, whose sole job is to ping the endpoint of the functions to keep them warmed up and cached in memory. As the plot shows, the time taken to insert the record into the Cosmos database is cut down drastically from the first run to the second run. The drop is from 5 seconds to about 1.5 seconds when it comes to insert the record. Similarly, the Azure Function responsible for generating the pdf also sees a drastic drop in the amount of time taken to generate and return the file. By testing the critical areas of the project, it further highlights the speed benefits provided over the current system where it can take a few minutes to receive the requested results.

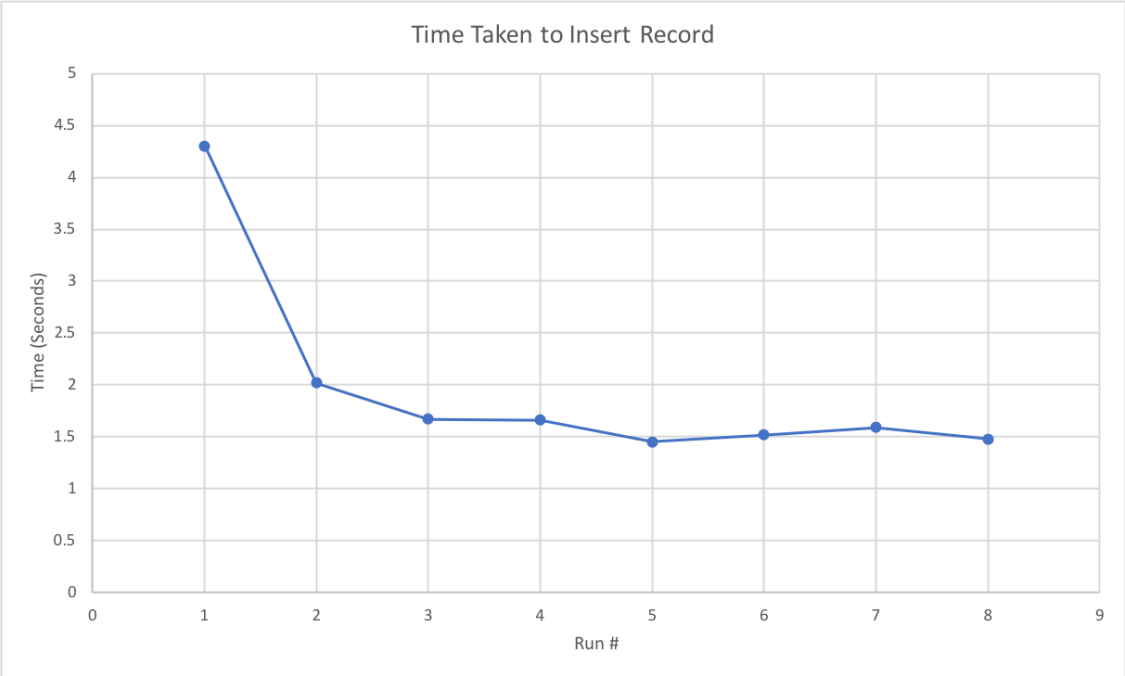

Figure 26 Time Taken to Insert Record

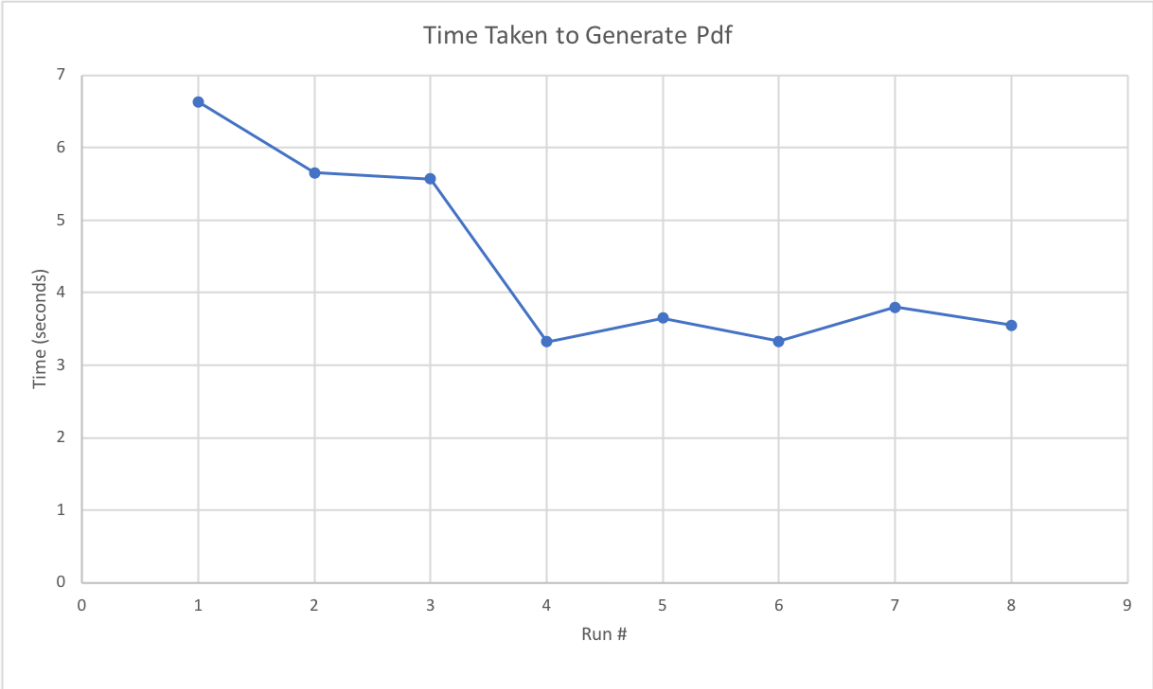

Figure 27 Time Taken to Generate Pdf

Chapter 8: Conclusion and Future Works

The desire to retake control over student's academic information led to the implementation of this project and guided it through the end. It was initially sought to implement a fully functioning blockchain for utilization in this project. However, that task proved to be too difficult for a single person to undertake in a reasonable amount of time. Thus, an alternative was pursued; the alternative contained the main sought-after functionality, which was the linking of records via cryptographically secure hashes. The goal was revised to implement someone that vaguely resembles a blockchain in concept, rather than a blockchain itself. There was also need to provide the user requesting the certificate with a website. Additionally, there had to be a tool that would allow the administrator of the records the ability to modify and verify the records.

The project started with the creation of the website first. The app had to simple and make a request to an Azure Function. Once the request was received by the Function, it would search the database, and prepare a pdf document with the student's information. The Function would then have to encode the pdf document as a Base64 string, so that it could be sent back as the response to the received request. The web application would then receive the response, extract the string, build the document blob and display it back to the user as a pdf document on their browser screen. The entire process also had to be reasonably fast. The process from start to end takes a few seconds on average.

After implementing the simplistic website, the first Azure Function responsible for generating the pdf document was implemented. It was tested with data that was inserted by hand into the Cosmos database emulator. The biggest drawback here was finding a library that would allow the creation of document and be able to be run in a

serverless environment with no external dependencies. Following that, various transmission encodings were researched until the Base64 was picked due to it being the perfect fit for the project. There was some difficulty encountered on the best way to display rebuild the binary and display it to the user, it too was solved after some research into the topic.

Lastly, there was a need for a tool that would allow for rapid testing of the creation of student record and the insertion of the data into the database. The tool generates a JSON file with the student's records and when the record is transmitted to the Azure Function responsible for the insert, calculates and handles the generation of hash values and the linking of nodes. One of the biggest functionalities delivered by the tool is the ability to verify and validate the authenticity of the entire chain of student records. It is responsible for computing the various required hashes and looking for a mismatch and then observing the user where the mismatch occurred.

The project provides completed workflows for the two different end users. The person requesting the data is able to get a verified certificate similar to the one generated by the current system. The person responsible for administrating the records is able to insert and verify the records. The entire workflow for either of the users is seamless and takes a few seconds on average, while simultaneously gaining all the benefits of a serverless architecture.

There are a few ideas that were thought of that could benefit from further improvement. The first area that could be further explored has to do with hybrid serverless platforms. It is possible that there could be further possible cost savings or unknown benefits resulting from the mixing and matching services from different

platforms. The compute service could be Azure Functions and, the web application can be hosted on an Amazon EC2 instance. It is unknown at this time what sort of advantages or disadvantages could result from such an arrangement. It would have to looked into for further detail.

An additional way the project can be further worked upon is by testing and implementing various strategies on handling a modified record chain. For the purposes of this project, it only went as far as determining if the chain of records was modified. To fix the issue, the user has to manually go to the failing record and fix the mismatch by hand. Another way to handle this issue would be to invalidate the entire chain and add the user records from scratch, thus rebuilding the entire chain. An alternative to the previous could be to only invalidate the remainder of the chain from the failing record. Alternatively, there could be a self-repair functionality that could be employed, where the failing record is restored back to the original values and the rest of the chain is repaired in a similar manner. Finding the correct implementation or the most effective method was beyond the scope of this project, as there would be issues unseen by a single person, and would require the feedback and input of everyone involved in the process.

Works Cited

[1] A. © C. S. University, N. 18111 N. Street, Northridge, and C. 91330 P.-1200 / C. Us, "Transcripts and Diplomas," *California State University, Northridge*, 08-Jul-2013. [Online]. Available: https://www.csun.edu/alumni/transcripts-and-diplomas. [Accessed: 10-Jan-2019].

[2] M. Fowler, "Serverless Architectures," *martinfowler.com*. [Online]. Available: https://martinfowler.com/articles/serverless.html. [Accessed: 10-Jan-2019].

[3] F. Bashir, "What is Serverless Architecture? What are its Pros and Cons?," *Hacker Noon*, 28-May-2018. [Online]. Available: https://hackernoon.com/what-is-serverless-architecture-what-are-its-pros-and-cons-cc4b804022e9. [Accessed: 12-Jan-2019].

[4] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, "Blockchain," *Bus. Inf. Syst. Eng.*, vol. 59, no. 3, pp. 183–187, Jun. 2017.

[5] "Blockchain explained," *Reuters*. [Online]. Available: http://graphics.reuters.com/TECHNOLOGY-BLOCKCHAIN/010070P11GN/index.html. [Accessed: 1-Feb-2019].

[6] S. W. 10 O. March, "Blockchain for Student Records," *Bold Business*, 10-Oct-2017. .

[7] "IBM unveils Blockchain as a Service based on open source Hyperledger Fabric technology," *TechCrunch*. .

[8] "About," *Learning Machine*. [Online]. Available: https://www.learningmachine.com/about. [Accessed: 23-Mar-2019].

[9] M. Techlabs, "What is Serverless Architecture? What are its criticisms and drawbacks?," *Medium*, 04-May-2017. .

[10] "Bare-metal server," *Wikipedia*. 28-Mar-2019.

[11] "A brief history of serverless (or, how I learned to stop worrying and start loving the cloud)," *freeCodeCamp.org News*, 05-Apr-2018. [Online]. Available: https://www.freecodecamp.org/news/a-brief-history-of-serverless-or-how-i-learned-to-stop-worrying-and-start-loving-the-cloud-7e2fc633310d/. [Accessed: 24-May-2019].

[12] May 26 and 2017, "Docker Downsides: Container Cons to Consider before Adopting Docker," *Channel Futures*, 26-May-2017. [Online]. Available: https://www.channelfutures.com/open-source/docker-downsides-container-cons-to-consider-before-adopting-docker. [Accessed: 24-May-2019].

[13] ggailey777, "Supported languages in Azure Functions." [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-functions/supported-languages. [Accessed: 26-Jun-2019].

[14] "AWS Lambda – FAQs," *Amazon Web Services, Inc.* [Online]. Available: https://aws.amazon.com/lambda/faqs/. [Accessed: 26-Jun-2019].

[15] "HTTP Methods GET vs POST." [Online]. Available: https://www.w3schools.com/tags/ref_httpmethods.asp. [Accessed: 26-Jun-2019].

[16] E. Geschwinde and H.-J. Schoenig, "An Introduction to SQL," 2002.

[17] S. Yegulalp, "What is NoSQL? NoSQL databases explained," *InfoWorld*, 07-Dec-2017. [Online]. Available: https://www.infoworld.com/article/3240644/what-is-nosql-nosql-databases-explained.html. [Accessed: 27-Jun-2019].

[18] timsander1, "Getting started with SQL queries in Azure Cosmos DB." [Online]. Available: https://docs.microsoft.com/en-us/azure/cosmos-db/sql-query-getting-started. [Accessed: 27-Jun-2019].

[19] “Hash function,” *Wikipedia*. 31-May-2019.

[20] “What Is Hashing? [Step-by-Step Guide-Under Hood Of Blockchain],” *Blockgeeks*, 06-Aug-2017. [Online]. Available: https://blockgeeks.com/guides/what-is-hashing/. [Accessed: 27-Jun-2019].

[21] “bcrypt,” *Wikipedia*. 05-Apr-2019.

[22] “Learn Cryptography - Hash Collision Attack.” [Online]. Available: https://learncryptography.com/hash-functions/hash-collision-attack. [Accessed: 27-Jun-2019].

[23] “Rainbow table,” *Wikipedia*. 13-Jul-2019.

[24] “appsec - How to securely hash passwords?,” *Information Security Stack Exchange*. [Online]. Available: https://security.stackexchange.com/questions/211/how-to-securely-hash-passwords. [Accessed: 18-Jul-2019].

[25] R. Dvojmoč, *C# .NET Core wrapper for wkhtmltopdf library that uses Webkit engine to convert HTML pages to PDF.: rdvojmoc/DinkToPdf*. 2019.

[26] “Wayback Machine,” 26-May-2013. [Online]. Available: https://web.archive.org/web/20130526224224/http://csrc.nist.gov/groups/STM/cavp/documents/shs/sha256-384-512.pdf. [Accessed: 21-Jul-2019].

[27] *Powerful .NET library for benchmarking. Contribute to dotnet/BenchmarkDotNet development by creating an account on GitHub*. .NET Foundation, 2019.

[28] J. Newton-King, *Json.NET is a popular high-performance JSON framework for .NET: JamesNK/Newtonsoft.Json*. 2019.

[29] “Postman | API Development Environment,” *Postman*. [Online]. Available: https://www.getpostman.com. [Accessed: 21-Jul-2019].

[30] “Genesis block - Bitcoin Wiki.” [Online]. Available: https://en.bitcoin.it/wiki/Genesis_block. [Accessed: 23-Jul-2019].

[31] “Key stretching,” *Wikipedia*. 13-Feb-2019.

[32] ggailey777, “Azure Functions scale and hosting.” [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-functions/functions-scale. [Accessed: 24-Jul-2019].