**CLOUD STORAGE**

****

**FACULTY OF SCIENCE AND TECHNOLOGY**

**MODULE: HCSE225 – MINI PROJECT**

**REG: R199147H**

**NAME: Ryan Ben**

Chapter 1: Prelude

One thing to note is that I am a very big fan of google firebase (<https://firebase.google.com/>), my favorite part being the fact that it is free to use : ) However there are other factors like it’s very easy to use, hence you do not need to know a lot about backend to build an entire application. Which I have done so by building 2 of my applications on it, Solve it (<https://github.com/rbryanben/Solve-It>) and ChatSock (<https://github.com/rbryanben/ChatSock-v1.0.2>). Bad choice to use Firebase with a C# application by the way!

I mostly use Firebase Authentication, RTD and Storage for my applications, and one thing I have noticed is that they offer the least security possible when it comes to files. Files are created with a n char long string so that a person may not easily guess the path of the file, but if the person is authenticated, they can view any file you created, they just must guess the URL.

So, my goal is to remake a service like Firestore, but this time be a little bit better than the engineers at Google and add secure access control.

Chapter 2: Introduction

“I will call it Winterstore.”

I usually startup by mocking up some UI designs before anything else, but here I ought to do things differently. My new starting point was to come up with a name for the service.

On my way back home after I was done with my level 2.1 examinations, it hit me…Winterstore. Store being the storage part, and hence the two words combined meaning something u would want to keep safe over the winter season.

Next was a logo and the color theme for the service, so I hit up my friend [Munya](https://zw.linkedin.com/in/munyaradzi-chigangawa-45170818b) (available on LinkedIn) to do his magic and come up with something I could use for my service, cause I really suck at graphic design. Of which he delivered way more than I expected by doing the following work:



Chapter 3: Research

“Down memory lane.”

The idea of building a cloud service was not new, it was always part of the plan to build something as big as firebase for my mini project, despite not having an idea of where to start.

I remembered reading a blog by Rio Weber [1] on remaking firebase real-time database, which is something like what I am making. On his blog he includes a statement by one of his friends who is an engineer at Firebase, which notes [1]

***“Specifically talking about Firebase Hosting, we do have a number of Node.JS servers running behind a load balancer (which is in turn behind our CDN) that store no data and maintain no state. All of the binary data is stored in a simple, central BLOB store and all of the settings and changing state is stored in a Firebase (like the version history, custom domains, rewrites / redirects etc.). This allows us to add / remove the Node.JS boxes as the load requires.***

***The Firebase Realtime servers themselves are all written in Scala, so are running on the JVM. We make extensive use of the Netty and Akka libraries, and the actual disk persistence on the servers has been abstracted to a simple key-value store.***

***Chris***

***Engineer @ Firebase”***

Although this answered some questions in the setup of the service, there were still questions that needed to be answering, which were.

*[QN 1] How will I store files larger than 10mb because SQL will consume a lot of resources?*

*[QN 2] How will I allow multiple people to use the service? With each person having their own files and folders*

*[QN 3] How will I bring the service to mobile applications and desktop application rather than web?*

So, I did some digging trying to find anything I could use, but the solutions to be related to the technologies I had existing knowledge in. My technologies which were Django .NET C# JAVA SQL and Mongo

Some of the solutions I found out to my questions where:

*[QN 1 Sol] Django Private Storage by [2] Edoburu (GitHub* [*link*](https://github.com/edoburu/django-private-storage)*), which allowed me to store files. However, this was down using [3]* [*Django FileSystemStorage class*](https://docs.djangoproject.com/en/3.2/ref/files/storage/)*, meaning it was not good for serving multiple people.*

*[QN 2 Sol] There was a point in time I was researching stacks used by companies, and I found this image of the stacks used by Instagram which implemented both a NoSQL database and a SQL database. Hence, why not use SQL for the logic and constraints of the service and store the files on some other database.*

*[QN 3 Sol]* [*Hacking Life*](https://stackoverflow.com/questions/12763792/how-does-instagram-use-django) *[4], on stack overflow answering how Instagram uses Django as their webserver states:*

***“IOS is the "Frontend" of the application. Django is a backend "Server Side" Web Framework. For example, if there is an iPhone application that shows you all the latest tutorial videos about baking (let’s say you had a website bakingtutorials.com), you would make an iPhone application in objective-c to run as the frontend. In your objective-c iPhone code, you would communicate with a "server" which runs Django/Python. The Server Side (Django) holds the video and other baking data in a database. The Frontend iOS application requests data from the Django application. Client Side & Server-Side communication. This is usually accomplished with a RESTful API - which in basic terms, means the iPhone application uses human-readable URLs to grab data from Django/the server's database. This data is sent to the frontend from the server where it is parsed by objective-c and formatted for the application.”***

With these questions answered I was already visualizing the whole system in my head. It became an obsession.

Chapter 3: System Summary

“The bigger picture.”

Here is a summary of the bigger picture. On the frontend side I will have my libraries and the web application which will be communicating with the backend via API and Channels. Meaning my Web Application will not just be following an MVC model, but MVVM as well.

The backend will have Django as my webserver, dealing with all the logic handling. Hosted on a UNIX socket by NGINX and GUNICORN as my gateway. Daphne will be used to handle all socket connections to the webserver.

Files will be store in Mongo GridFS, which allows me to stream a file instead of loading it into main memory. Lastly, I will be using MySQL to store all data that needs constraining.

Diagram

Description automatically generated

Chapter 4: Backend

“Ohh hail python.”