TLDWorkerBee

Team: Austen Christensen, Morgan Lovato, Wei Song
Sponsor: Harlan Mitchell - Honeywell
Mentor: Austin Sanders

Technological Feasibility Analysis
November 7, 2018

Table of Contents

1. Introduction

Planes are one of the most popular ways to travel quickly. They can get from Phoenix to Los Angeles in a little over an hour. These machines generally have a cruising altitude of 33,000 to 42,000 feet, making it vital to have working engines at all time. Our team is TLD Worker Bee, and we are working on the project Prototype Time Limited Dispatch (TLD) Application. Our sponsor is Harlan Mitchell from Honeywell Aerospace, a manufacturer of aircraft engines. The current way of getting TLD data is to physically connect a computer to the engine computer via USB and download the data. This can be very cost inefficient because the mechanic has to be on site to gather the data, then determine what work needs to be done on the engine. They might not have the parts or tools they need, making the plane stay grounded longer. However, our sponsor has found a way to wirelessly upload that data from the plane onto a cloud. Our job is to build a prototype GUI that will quickly and accurately display the data that has been uploaded to the cloud. This will make it so a mechanic doesn't have to be physically next to the plane to diagnose the error codes.

This feasibility analysis is structured to help the team choose a solution to the key technological challenges of this project. We are identifying possible solutions, researching them, and selecting a solution based on an analysis of each. We begin by identifying the different technical challenges we have found for this project. Then, in the subsequent sections, we analyze each challenge individually; in this individual analysis, we compare three different alternative solutions to the challenges and choose the best one. We then explain how all of these solution will be integrated together to help us complete this project.

2. Challenges Overview

Before we dive into the analysis, we must present an overview of our challenges. This project presents us with four major technological challenges. Our first challenge is finding a way to replicate the existing cloud environment Honeywell is currently using to store their Time Limited Dispatch (TLD) data. We need to have our own cloud environment where we can upload TLD test data for the purpose of our project. This cloud must be able to take in a file of data, make sure the data does not get compromised, and allow that data to be downloaded into a database.

That brings us to our next challenge: we need a database to store this TLD data once we get it out of the cloud. There is a possibility that our database must be stored in the cloud as well as stored on a local host. The cloud version of the database would be used as an "original" while the local database will be copies of the original. The database must then be viewed in some sort of GUI.

The GUI for this challenge is another technical challenge we are facing. There are so many kinds of GUI to display the TLD data, like web app, desktop application, command line, or even

readable file, and we need to choose the best way to evaluate and display those TLD data, and it has to work seemly with the cloud and database we chose.

Since our project will be reliant on the internet, there must be an internet protocol to follow. Our project will need to be able to connect to a remote server to retrieve data off of the cloud. It needs to be a reliable transfer of data and the data must be exactly the same as in the cloud, in order and accurate.

3. Cloud

3.1. Overview

There is already an existing workflow for the basis of this project. TLD data in a file is uploaded to a cloud, and the data is then displayed in a chosen GUI without any errors in the data. One technical challenge of this project is simulating or recreating the cloud environment our sponsor is currently employing. The solution we come up with to recreate a cloud environment must be able to take in a file of data and make it downloadable while ensuring the integrity of the data throughout the process.

The issue of needing to use or simulate our own cloud can be solved in various ways. For businesses, cloud computing allows for convenient access to offsite computing resources and storage without having the physical infrastructures to facilitate the necessary computing. For smaller organizations, it may be cheaper to consider simulating the cloud environment; this is accomplished with a cloud simulator. A cloud simulator allows for the modeling of different cloud applications and can be an alternative to making a physical cloud. Some clouds offer processing powers, some clouds offer storage capabilities, and some clouds offer both. What we need for this project is a cloud we can upload to and download from that will serve as storage for test TLD data files. Preferably, this cloud will have support for Python integration and implementation to allow consistency of the project and provide a free to use solution for the duration of this project. A possible limitation we may be faced with is the amount of available storage the solution may offer.

3.2. Alternatives

For this project, our team can either use an existing cloud service or simulate our own. We will be considering three alternative to this solution: Amazon Web Services, Google Cloud, and CloudSim. Both Amazon Web Services and Google Cloud are cloud services and CloudSim is a cloud simulator. Each alternative option will be evaluated against the following criteria:

- if it meets the goal - we need a solution capable of storage capabilities
- the language - the supported languages may affect the solution’s integration

- the cost or availability of the service - we need something free preferably
- available storage - we need a solution capable of storing the necessary data
- community outreach - this helps with getting help and how well maintained the solution is by the provider

3.2.1. Amazon Web Services

AWS is a cloud services platform with many services and security capabilities. A few main services offered are computing power, storage options, networking and databases, content delivery, and other. There are over 50 services available in total.

The AWS services we are interested in are the Cloud Storage services which are described as a reliable and secure place to store data. There are four different AWS Cloud Storage products to choose from, each with their own key functionalities. Simply defined, each product is:

- Amazon Elastic Block Store (Amazon EBS) – for storing and processing block data like relational and NoSQL databases and Big Data processing
- Amazon Elastic File System (Amazon EFS) – for storing and sharing data in scalable file systems
- Amazon Simple Storage Service (Amazon S3) – for storing and accessing any type of data from any internet location
- Amazon Glacier – for low-cost and long-term archive of data

Based on the needs of this project, we would be using the Amazon S3. It is both secure and 99.999999999% durable meaning there is an annual average expected data loss of 0.000000001%. With this product, we can store and retrieve data in a cloud.

There are many different platforms which AWS can run on including Android, iOS, Java, PHP, Python, Ruby, and more. The team is hoping to complete this project using Python as the main language, and there is an AWS SDK for Python called Boto3. The SDK works with Amazon S3 as well. The team could also install Python onto the cloud server to run scripts.

AWS offers a Free Tier that includes a 12-month trial of Amazon S3. The trial includes 5GB of storage, 20,000 Get Requests, and 2,000 Put Requests. 12 months starting now (in November) is more than enough for this service to last the entirety of this project.

Since the services are provided by Amazon, which is a large company, new services are quickly built to respond to changing requirements in business, and there are many strong security features in place for user protection and data encryption. The AWS cloud itself has a large network of regions that spans the

globe; there are 55 Availability Zones in 18 different geographical regions with three of these zones being in the Northern California region. There are also many well known companies, such as Spotify and Yelp, that use AWS as well and many that even use a Free Tier.

One major downside or thing to consider when it comes to using AWS is the learning curve involved.

3.2.2. Google Cloud Platform

Google Cloud is another secure cloud services platform. It offers services like computing, AI and machine learning, storage, data analytics, and more.

The Google Cloud services we are interested in are the Cloud Storage services. The storage services are secure and durable that allows instant access to the data. This will work for our project as we need to store and access data.

Google Cloud supports such languages as C#, Java, Python, PHP, Ruby, and more. Our project is hoped to be completed in all Python, and Google Cloud does have support for Python.

There is a free trial of the Google Cloud Platform services that includes access to all Google Cloud Platform products and $300 credit to spend over the 12 months of the free trial. The Always Free usage limits include 5BG of storage a month, 5,000 Class A operations, and 50,000 Class B operations. There are three categories of operations: Class A, Class B, and free. Class A and B operations are actions that make changes to or retrieve information in the cloud storage; they have different rates per 10,000 operations, but the Always Free trial allows a certain amount of operations to be done for free.

Google is another large company that can offer full support services and has one of the highest security ratings that can be given. The Google cloud itself has 55 zones in 18 regions around the world regions with three of these zones being in the Los Angeles region. Many well known companies including Blue Apron and Johnson & Johnson use Google Cloud services.

One downside to the Google Cloud Platform is the slow pace of improvements compared to other option being analyzed.

3.2.3. CloudSim

CloudSim is a library developed at the University of Melbourne in Australia to simulate cloud computing. It can simulate data centers, virtual machines, applications, and more. CloudSim is an open source project that is free to use.

It is possible to use CloudSim to simulate data centers we could use for this projects purpose of data storage, but it is an extensive process.

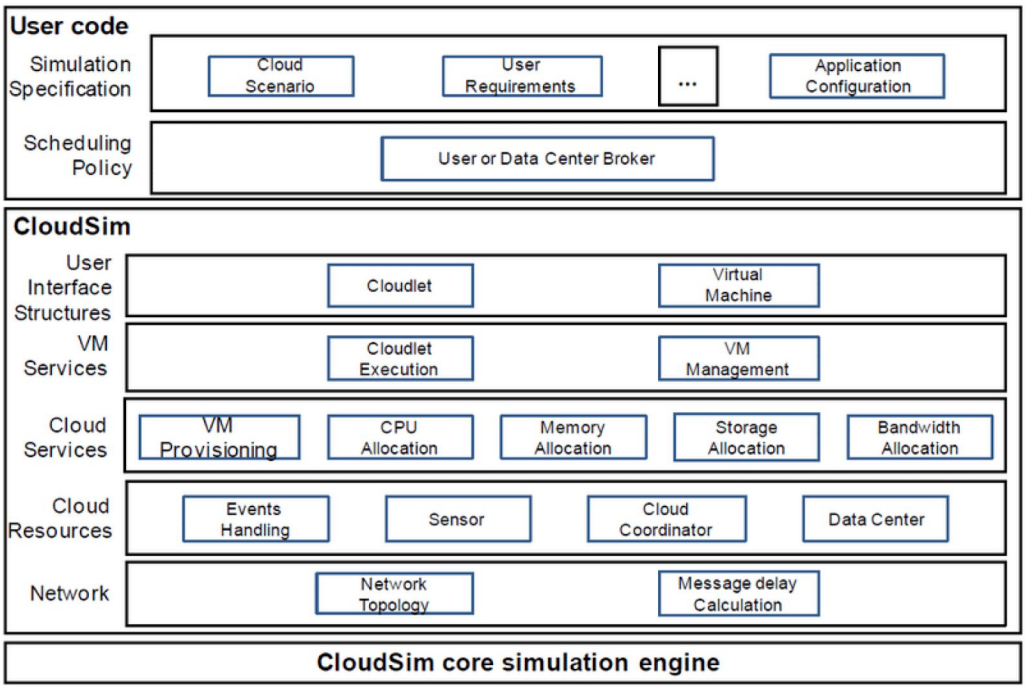

Fig. 1. CloudSim Architecture (featured in the article “A Features-based Comparative Study of the State-of-the-Art Cloud Computing Simulators and Future Directions” in the *International Journal of Advanced Computer Science and Applications*)

CloudSim requires a written Java program to compose scenarios and collect results from the Cloud applications. There is a detailed architecture of CloudSim pictured above. Each layer of the architectures communicates through message passing. Data centers are located in the second layer under Cloud Resources.

There are many tutorials, forums, and groups online where one can go to learn, discuss, and troubleshoot issues to do with CloudSim. From their website, there is a mailing list to sign up for that serves as a discussion group as well. There is also documentation for an ioline course on CloudSim, examples, and a README.

The major downside to CloudSim is there is no GUI.

3.3. Chosen approach

Table 1 compares the different chosen cloud models described in this section for different necessary functions including:

Table 1 - Cloud Comparison

Functions	**Amazon Web Services**	**Google Cloud**	**CloudSim**
Goal (Storage)	Met	Met	Met
Language	Many (Python)	Many (Python)	Java
Cost/Availability	Free Tier (1 year)	Free Trial (1 year)	Open Source
Available Storage	5GB	5GB/month	Undetermined
Community Outreach	Extensive - Responsive	Extensive - Slow	Limited

For this project, we will be using Amazon Web Services. We want to stay away from cloud simulation and use cloud services provided by a company. We chose this solution because it is the most practical and flexible for our needs. It has a free tier and can also be implemented in Python which integrates well with the other aspects of our project. The responsive community makes this choice more attractive than Google Cloud.

3.4. Proving feasibility

The way Amazon S3 stores data is as objects in buckets. Objects are a file and any metadata for that file and buckets are containers for objects. Uploading a file is how you store an object in a bucket.

AWS offers a Getting Started Guide for Amazon S3 on their website. The steps are as follows:

- Sign Up for Amazon S3
- Create a Bucket
- Add an Object to a Bucket
- View an Object
- Move an Object
- Delete an Object and Bucket

I plan on following this guide to learn the basics of how to use Amazon S3. A possible demo I could develop is how to add and delete a file to a bucket.

4. Database

4.1. Overview

This project will require a database to read and store all of the TLD data from the TLD file. There is a possibility for the database to be in the cloud for testing so the database platform must work well inside a cloud as well as on a host computer. The fields of the data will stay consistent from file to file but the data will change.

The issue of needing to use a cloud based database or a local database can be solved in a few different ways. What we need for this project is a database that will serve as storage for the TLD data files. There is a possibility that our project will require a cloud based database for testing. Preferably, this database will have support for Python integration and implementation to allow consistency of the project and provide a free to use solution for the duration of this project. A possible limitation we may be faced with is the amount of available storage the solution may offer as well as how the data is stored.

4.2. Alternatives

For this project, our team can either use a database designed for the cloud and/or a host computer. We will be considering three alternative to this solution: Amazon Relational Database Server(RDS), MongoDB, and Apache Cassandra. Both Apache Cassandra and MongoDB are NoSQL databases and Amazon RDS uses SQL. Each alternative option will be evaluated against the following criteria:

- if it meets the goal of storing the data correctly
- the supported language
- the cost to use the database and how much we can store
- the type of database
- how the database is scaled

4.2.1. MongoDB

MongoDB is a document-oriented database using JSON like format. The fields can vary from document to document and data structure can be changed at any point in time. A few of MongoDB's main features are Ad hoc queries, indexing, replication, load balancing, file storage, aggregation, and transactions.

The feature that draws the most attention to MongoDB is how it stores data. They claim to be the most flexible database using JSON-like documents. This means that fields can vary from document to document and data structure can be

changed over time. The document model maps to the objects in application code, making data easy to work with. Ad hoc queries, indexing, and real time aggregation provide powerful ways to access and analyze the data. MongoDB is a distributed database at its core, so high availability, horizontal scaling, and geographic distribution are built into the system and are easy to use. While MongoDB may be the most popular option, our data will be staying consistent and we don't need the ability to have multiple fields in different documents. Table 2 displays the pros and cons for MongoDB.

Table 2 - Pros and Cons of MongoDB

Pros	Cons
Speed	Documents can have different fields
Schema-less	Larger Data Size
Choose level of consistency	Less flexible with querying
	Only recent support of transactions

4.2.2. Apache Cassandra

Apache Cassandra is a free open-source, NoSQL distributed database. Since Cassandra is a distributed database, the data can be stored in multiple computers on a network of interconnected computers. A few of Cassandra's main features are offering CQL (Cassandra Query Language) and fault-tolerance.

Apache Cassandra database excels in scalability and high availability without compromising performance. Apache Cassandra provides lower latency for users and can survive regional outages. Cassandra Query Language is Apache Cassandra's interface alternative to SQL. CQL adds an abstraction layer to hide implementation details and provides native syntaxes for collections. There is a CQL driver for Python to make it easier to use and seamlessly connect to other technologies. Table 3 displays the pros and cons for Apache Cassandra.

Table 3 - Pros and Cons of Apache Cassandra

Pros	Cons
SQL like language using CQL	Unpredictable performance
Multi-Data Center replication	JVM required to store huge amounts of data
Write speed	

4.2.3. Amazon RDS

Amazon RDS is a distributed relational database service by Amazon Web Services. A few of Amazon RDS' main features are Multi-Availability Zone (AZ) Deployment, read replicas, performance metrics and monitoring, and automatic backups. Amazon RDS has multiple database engines to choose from including MySQL, MariaDB and Microsoft SQL Server.

Since Amazon RDS can run on MySQL, it has some of the same benefits that MySQL has to offer. MySQL is an open-source relational database management system that is developed by the Oracle Corporation. A few of MySQL's main features are cross-platform support, full-text searching and indexing, and query caching. On top of that, Amazon RDS makes it easier to set up, operate, and scale in the cloud with added benefits of cost-efficiency and resizable capacity while automating administration tasks such as backups and hardware provisioning. This makes it so the developers can focus on the applications for fast performance, high availability, security and compatibility. Table 4 displays the pros and cons for Amazon RDS.

Table 4 - Pros and Cons of Amazon RDS

Pros	Cons
Easy adding/removing replicas Managed Service	Size limit Hard to tune

4.3. Chosen approach

Table 5 compares the different chosen database structures described in this section for different necessary functions including:

Table 5 - Database Comparison

Functions	**MongoDB**	**Apache Cassandra**	**Amazon RDS**
Language Support	Many(Python)	Many(Python)	Many(Python)
Query Language	NoSQL	CQL	SQL
Cost/Availability	Open Source Free up to 512MB	Open Source Requires JVM	Free Tier (1 year) up to 20GB
Type	Document-oriented	Distributed	Relational
Scaling	Horizontal	Horizontal	Vertical

For this project, we will be using Amazon RDS. We want to stay away from having to pay for database storage as well as be able to store an unknown amount of data in it. We chose this solution because it is the most practical and flexible for our needs. It will connect seamlessly to AWS and it works well with Python. Since our database also has the possibility of living in the cloud as well as on a computer, we wanted a more cloud-based platform. We can use Amazon RDS as our cloud database and MySQL for a local instance.

4.4. Proving feasibility

The way MySQL stores data is in rows. Each new “piece” of data gets stored in a single row with a key that groups the data together. All new “pieces” of data must have the same number, type, and title of fields in order to be stored into that database. This is really helpful for us because all of our data should stay consistent and the fields shouldn’t vary from data entry to data entry. A working demo would be simply storing our TLD test data into the database on both MySQL and Amazon RDS. Change the data on one of them, then try to push data from there onto the other database to see if the new database changes.

5. GUI

5.1. Overview

Another technical challenge for this project is to choose the appropriate GUI to display the data. According to the project description, this project is to prototype a modern cloud-connected GUI for the purpose of evaluating TLD data, so a simple GUI to display the correct data is necessary. When we communicate with our sponsor, we come to a consensus that there could be many possibilities or option for the GUIs. It can be any

kind of forms such as web app, desktop software, command line or even files that can be opened and read the data directly. It doesn't need to be very complicated or fancy, as long as we can maintain the data integrity and display the correct data only. However, we still need to find out one GUI among these options that provide an outstanding readability and simplicity.

Since this project is a web-based prototype, which includes lots of data transportations on the internet, we will mainly focus on the web app or web framework, which can display the data in an organized way through our browser. We will also look into the other kinds of GUIs such as desktop applications (based on different platforms) and even command line or readable files.

5.2. Alternatives

For the web app, we have some options among some established frameworks based on different programming language. Preferably, we would like the framework to support the Python implementation to maintain the consistency of the project. According to the statistics from "Python's Web Framework Benchmarks" The top 3 popular choices for web app frameworks are Django, flask and Tornado, and all of them are based on Python. We will evaluate these three frameworks based on the following criteria:

- Performance (serialization performance, remote performance, and database performance)
- Functionality
- Simplicity (for the implementation)
- Support Documentation
- Open Source
- Portability

5.2.1. Django

Django is a free and open-source web framework based on Python environment. It follows the model-view-template (MVT) architectural pattern. It provides a simple way to develop the complex, database-driven website. Django emphasizes reusability and pluggability of components, which is actually a very advanced idea: the components are removable. If you don't want a component of the application, you can delete it directly without affecting other components in the system.

Compared to other frameworks, Django provides a very completed solution. It has assigned all the component like template, Session, authentication and even app partition. For example, We can develop a powerful backend with its admin and Django-suit. It provides as much as functionality, which is very suitable for

the development of enterprise level software. And it also has a killer feature, that is its well-known and powerful data management component. With its ORM component, a novice can easily access and operate the database without learning SQL language. It comes with very detail debug information, which is very easy to find out the error in the code.

However, there are also some disadvantages to this framework. Django includes too many functional components for a lightweight application, even if they don't actually need it. It also has an over-encapsulation problem. Many classes and methods are encapsulated, which are very easy to use but very difficult to modify. Also, compared to other platform based on C & C++, the performance of Django is low. However, that is a common problems for most of the Python-based frameworks. The flexibility for implementation is just so-so. The Django template separates the code and style and doesn't allow Python code to appear in the template, which may not be sufficient for some programmers.

5.2.2. Flask

Flask is another popular web framework written in Python. Applications implemented with Flask framework include Pinterest and LinkedIn. Developers who switched from Django to the Flask will certainly agree with that. It does not require particular tools or libraries, and it has no database abstraction layer or any other pre-existing components. However, it supports extensions that can add application features as if they were implemented in Flask itself. It has a wide selection of third-party libraries, which is free, flexible and extensible.

Also, it's easy to get started, even if the developers don't have much experience in web development. It's very easy to develop API for web services and also very suitable to develop a small website. It might be difficult to develop large websites using this framework, since all the code architecture needs to be re-designed, which depends on the developer's ability and experience.

For the data management, Flask is commonly used with MongoDB, which gives it more control over databases and history. It might not be work with relational databases as well as Django, but it works much better with NoSQL databases. For the performance, Flask shows an advantage in local JSON serialization. The average time for Flask to complete a JSON serialization was 43.33 milliseconds and 4630 requests per second, which is comparable to Django with 42.52 milliseconds complete time and 4762 requests per second.

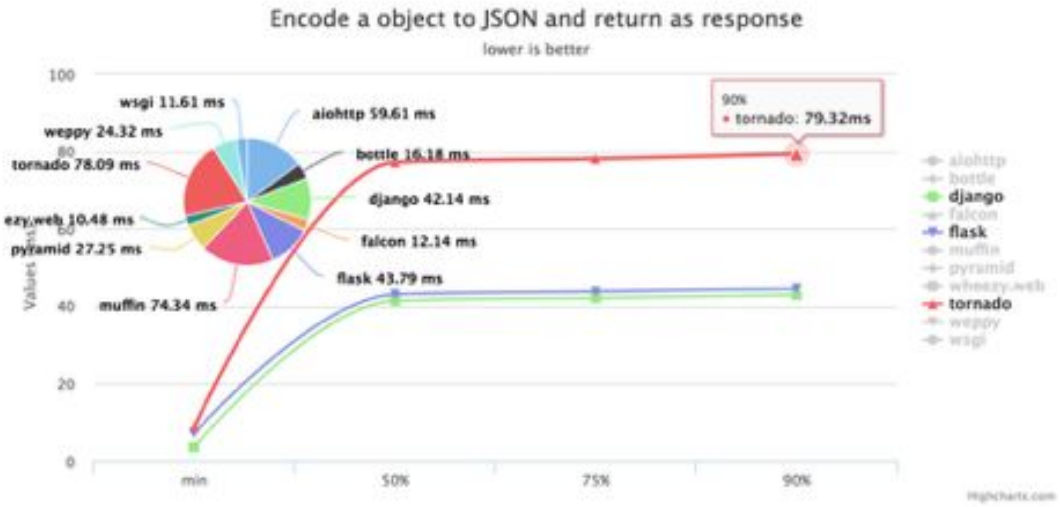

Fig. 2. Serialization Performance Comparison (JSON encoding)

5.2.3. Tornado

Tornado is a scalable, non-blocking web server and web application framework written in Python. Like Flask, Tornado also tries to keep the framework simple and powerful. It only provides some necessary component like URL Router and very basic database access. It doesn't encourage the developer to use templates, but unlike Django, it allows the developers to embed single line python code in the template, which offers some limit flexibility.

Tornado is noted for its high performance. It tries to solve the C10k problem affecting other servers, allow them to scale beyond 10,000 connections or clients. It's also very good at remote processing performance. According to a performance test, the average time for Tornado to load and return http response from a remote serve is only 1.05 seconds, while Flask is 3.34 seconds and Django is 3.48 seconds. The HTTP response speed of Tornado is three times faster than Flask and Django. Tornado's success is due to its inherent asynchronous nature, while Django and Flask are synchronous frameworks with limited performance when processing the requests. If taking both the corresponding HTTP speed and the JSON processing speed showing in 5.2.2 into consideration, which will show the average performance of data processing through the internet, Tornado takes 1114.48 ms, which is still 3 times faster than Django (3519.88 ms) and Flask (3387.60 ms).

However, for the database support, it only provides a very basic encapsulation of MySQL. And due to its single-threaded feature, which means that if the database query returns too slowly, the entire system response will be blocked. It doesn't have a powerful database management system like the ORM in Django, and it

doesn't even support the session. But it also provides an opportunity to improve the database performance.

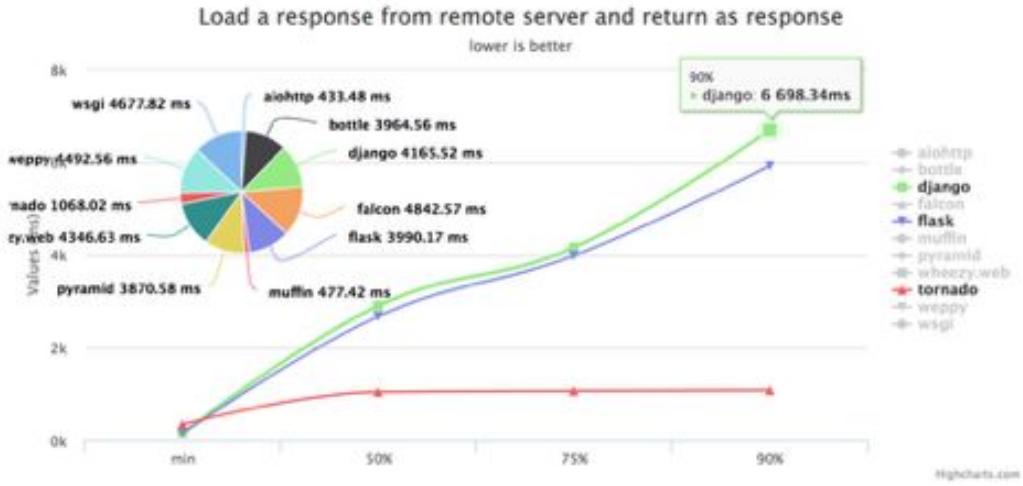

Fig. 3. Remote Performance Comparison (HTTP Response)

5.3. Chosen approach

Table 6 compares the frameworks of the different chosen GUIs described in this section for different necessary functions including:

Table 6 - Framework Comparison

Functions	**Django**	**Flask**	**Tornado**
General Performance	Medium	Fast	Very Fast
Language	Python	Python	Python
Open Source	BSD	BSD	Apache 2.0
Functionality	Powerful	Limited	Medium
Flexibility	Limited	Very good	Good
Simplicity	Complex	Simple	Medium
Portability	Good	Very Good	Limited
Support Documentation	Well-supported	Medium-supported	Medium-supported

Table 7 compares the performance of the different chosen GUIs described in this section for different necessary functions including:

Table 7 - Detailed performance comparison

Functions	**Django**	**Flask**	**Tornado**
Serialization Performance (JSON)	Fast (4762 pr / 42.52 ms)	Fast (4630 pr / 43.33ms)	Slow (2578 pr / 77.51 ms)
Remote Performance (HTTP response)	Slow (3.48s)	Slow (3.34s)	Fast (1.04s)
Database Performance	Slow (42.9 pr / 2904.4ms)	Fast (123pr / 1440.24ms)	Fast (143pr / 1344.69ms)

For this project, we will be using Django. It's the most popular web-app framework in Python, which means excellent document and community support. It integrated many powerful functions and components, which will simplify our way to implement this project. In some key performance indicator like database accessibility, it shows the advantages compares to other popular frameworks like Flask and Tornado, which is crucial for business users and also for the data-based project like this.

5.4. Proving feasibility

In order to prove the feasibility of Django, We will create a local sample database with the ORM model and transfer some sample TLD data into the database. A program written in Python will evaluate the integrity of those TLD data and eliminate the incorrect data. Only the correct data will show on the frontend using Template in Django.

6. Protocol

6.1. Overview

To transfer messages over the internet, the message must be associated with a protocol. There are only two types of transport level protocols and multiple application level protocols. Application level protocol determine how the two applications will work with each other while the transport layer protocol determines how those messages get from point a to point b.

6.2. Alternatives

There are currently two main types of protocols: TCP and UDP. The third protocol that we will be analyzing is the FAA approved Common Message Handling Protocol (CMHP).

6.2.1. TCP

TCP is the most used internet protocol out there because it is more reliable than UDP. When a packet is sent, there is a destination as well as a source address. If the receiver does not receive the packet, then the sender re-sends it. These added headers adds overhead for each router the packet gets to making the total trip time longer than if you used UDP.

Table 8 - Pros and Cons of TCP

Pros	Cons
Reliable Congestion control Flow Control	Slow

6.2.2. UDP

UDP is the fastest way to send messages on the internet. The only thing that UDP requires is a destination IP address. The reason that UDP is the fastest is because it doesn't offer any sort of reliability. The sender just sends out the packets without knowing what is on the network and if the receiver received those packets.

Table 9 - Pros and Cons of UDP

Pros	Cons
Fast	Unreliable

6.2.3. CMHP

Common Message Handling Protocol (CMHP) is an application-level protocol developed by the FAA. It was created because TCP/IP is a fire-and-forget protocol with no message receipt acknowledgment and little socket management. CMHP adds functionality that provides message-delivery assurance, regardless of the number of sockets that are traversed between end-systems. This protocol sits in the application layer at both the client and server side. CMHP enables the exchange of application data with assurance that the data sent has been successfully received and processed by the far-end application.

Table 10 - Pros and Cons of CMHP

Pros	Cons
Ensures data integrity FAA Approved Security	Have to contact organization for source code or create it ourselves

6.3. Chosen approach

Table 11 compares the different chosen protocols described in this section for different necessary functions including:

Table 11 - Protocol Comparison

Functions	**TCP**	**UDP**	**CMHP**
Security	Secure	Least Secure	Most Secure
Speed	Slowest	Fastest	Slowest
FAA	No	No	Yes

For our project, we will be using CMHP because it is even more secure than TCP and is approved by the FAA.

6.4. Proving feasibility

Since CMHP is an application-level protocol, it will work seamlessly with the rest of our project. All we have to do is add a few lines of code into our source code and CMHP will work on anything that uses the internet.

7. Integration

In the technology integration, we now need to consider how to connect all the components we have chosen above in order to fulfill the requirement of this project. For now, we have three main components in this project, they are:

1. Cloud (Amazon Web Services)
2. Database (Amazon RDS)
3. GUI (Django)

And all the data transfer are based on CMHP protocol we use the arrows to represent the transfer of data between components in the architecture graph below:

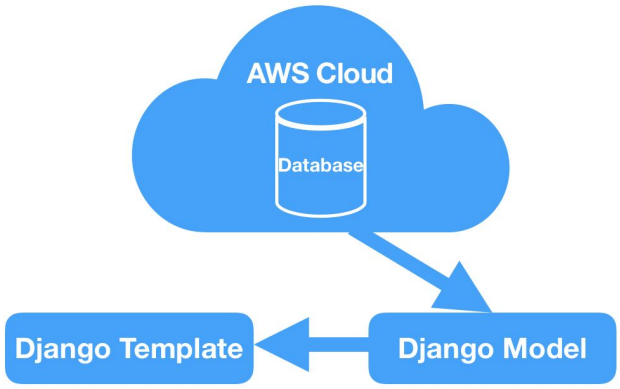

Fig. 4. Software Architecture

Since the CMHP protocol will ensure the data integrity during the data transfer process, then we can assume that the data download to the GUI is the same as in the Cloud. So in this architecture, all the data flows are unidirectional: from the Cloud(database) to the backend of the GUI through the CMHP protocol, then display in the front. We still need to evaluate the TLD data in the GUI so that the incorrect data cannot be displayed.

7.1 Transfer TLD data from the Cloud to the GUI

Since the TLD data are stored on the files, we need to transfer them to the database (Amazon RDS) in order to process them. We use the backend program written in Python to connect these two components and finish the transfer process. The database (Amazon RDS) need to have configurable paths in order to access it. Then we transfer those data to the GUI through the CMHP protocol, this will ensure the data integrity during the transfer process and also provide the authentication feature. When the GUI tries to connect and interact with the Cloud through HTTP requests, the correct credentials need to be used.

7.2 Evaluate and display the data

After the TLD data have been downloaded to the local GUI, we can evaluate those TLD data and eliminate the incorrect data. This could be finished within the ORM model in Django with a backend program written in Python and then display those correct data on the Frontend Website.

8. Conclusion

Our team was faced with four technical challenges, and we had to choose a solution for each one. To recreate a cloud, we chose to use Amazon Web Services S3 cloud storage services. For our database, we will be using Amazon RDS as our cloud database and MySQL for a local instance. Our GUI will be implemented using Django and we will use CMHP our message transferring protocol.

This document has been the team's technical feasibility analysis for this project. We have analyzed the challenges of this project, compared different solutions, and come to a decision on how we will solve the challenges we are faced with and how these solutions will integrate with each other for the final product.

Below is a final review of our findings. Table 12 shows the different technical challenges of this project, their chosen solutions, and the confidence level the team has in each chosen solution.

Table 12 - Chosen Solutions

Technical Challenge	**Chosen Solution**	**Confidence Level**
Cloud	Amazon Web Services	High
Database	Amazon RDS/MySQl	High
GUI	Django	High
Protocol	CMHP	High

Our next steps will be to being implementing and integrating the solutions we have chosen. Our chosen cloud solution will allows us to upload TLD test data and later download that data into our chosen database. We have chosen a GUI that will be both functional and practical for this project and the chosen protocol will allows us to ensure data integrity while complying to FAA standards.