A Web-Based Solution for Small Unmanned Aircraft System (sUAS) Traffic Management

Monetta Angelique Shaw
University of Nevada, Las Vegas, shawm11@unlv.nevada.edu

A WEB-BASED SOLUTION FOR SMALL UNMANNED AIRCRAFT SYSTEM (SUAS) TRAFFIC MANAGEMENT

By

Monetta Shaw

Bachelors of Science – Computer Science
University of Nevada, Las Vegas
2013

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
May 2016

UNLV | GRADUATE COLLEGE

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

April 26, 2016

This thesis prepared by

Monetta Shaw

entitled

A Web-Based Solution for Small Unmanned Aircraft System (SUAS) Traffic Management

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
Department of Computer Science

Ju-Yeon Jo, Ph.D.
Examination Committee Chair

Ajoy Datta, Ph.D.
Examination Committee Member

Yoohwan Kim, Ph.D.
Examination Committee Member

Emma Regentova, Ph.D.
Graduate College Faculty Representative

Kathryn Hausbeck Korgan, Ph.D.
Graduate College Interim Dean

Abstract

A Web-Based Solution for Small Unmanned Aircraft System (sUAS) Traffic Management

By

Monetta Shaw

Dr. Ju-Yeon Jo, Examination Committee Chair

Associate Professor, Department of Computer Science

University of Nevada, Las Vegas

There is currently no system in place for safely managing small unmanned aircraft systems (UAS). Small UAS (sUAS) pose a safety hazard to both manned aircraft and people on the ground. The initial solution to this problem proposed in this paper is a system that consists of a web server and an Android mobile application (app). In this solution, the server is only accessible through the Android mobile app. After evaluating the initial solution, we determine that the system in this solution would be unable to quickly adjust to changes in technology and in the sUAS industry. To solve this shortcoming, the second version of the solution has the server host a web service that allows any software or device that can use the Internet to utilize the server's sUAS traffic management (sUTM) functions through an application programming interface (API). In this solution, the Android mobile app we develop serves primarily as an example of software using the server through the API. After designing, implementing, and testing both the server's sUTM functions and the Android mobile app, we evaluate the solution again and identify the solution has limited ability to attract and retain users (sUAS operators). To lessen this shortcoming, the third version of the solution includes design elements commonly found in video games to help attract more users and make sure they continue using the sUTM system.

Acknowledgements

I would like to sincerely thank my Advisory Committee Chair, Dr. Ju-Yeon Jo, for her kindness and patience during my thesis. Her guidance was invaluable.

I wish to extend my sincere appreciation to Dr. Yoohwan Kim for his leadership, instruction, and recommendations. His advice given to me during my research project was immeasurable.

I would like to thank my parents, James Oscar Shaw and Mary A. Shaw for their consistent and continual support, encouragement, and direction throughout my life.

Thank you to my younger brother, James Oscar Shaw II, for always believing in me.

Dr. Emma Regentova, thank you for being one of my Advisory Committee Members.

I would like to especially thank the graduate coordinator, Dr. Ajoy K. Datta, for his counseling throughout my Master Degree Program.

Table of Contents

List of Tables

List of Figures

Chapter 1: Motivation

In the U.S., the Federal Aviation Administration (FAA) said that one of the safety concerns with unmanned aircraft systems (UAS) (commonly known as *drones*) is the ability for the aircraft to “see and avoid” other aircraft. [1] Because the operator and the unmanned aircraft are separated, the “see and avoid” capability of the aircraft is reduced when compared to that of manned aircraft. [1] This reduced “see and avoid” capability of UAS endangers nearby manned aircraft, people, and animals. [1] The radar-based technology currently used by Air Traffic Control to manage the air traffic is not effective for small UAS (sUAS) because their small size makes it difficult to detect them by radar. [2] Most sUAS, similar to the ones shown in Figure 1, Figure 2, and Figure 3, can easily be purchased online or at a local store, and controlled easily without any rigorous training or certification. Because of this, many sUAS operators are inexperienced with operating an sUAS and do not know many of the FAA regulations concerning sUAS operations and sUAS operation best-practices. Even if an sUAS operator knows the FAA regulations and the sUAS operation best-practices, obtaining the information needed to operate an sUAS safely and responsibly is difficult and cumbersome. If the current state of sUAS management remains for too long, many people will be injured or killed in accidents caused by unsafe sUAS operation.

Figure 1 Parrot Bebop 2. Image Source: [3]

Figure 2 DJI Phantom 4. Image Source: [4]

Figure 3 3D Robotics (3DR) Solo. Image Source: [5]

1.1 Required Features

Before we can evaluate any solution to the problem of sUAS traffic management, we must first know what an ideal solution is. The following is a list of criteria to use to evaluate a solution and describes an ideal solution.

- **Maintainability**: Here are the factors that should be considered: (1) Is it easy to modify and adjust components of the system? (2) How many people are needed to keep the system functional? (3) How many people are required to maintain and fix system issues? (4)How much time will be needed maintain the system?

An ideal solution is a system that would allow it components to be easily modified and adjusted as needed without taking much time or human effort. An ideal solution would also require little or no human intervention to keep the system functional. As a result, the system would require only a few people to keep the system functional.

- **Adaptability**: Can the system adjust to changes in technology, laws, and the sUAS industry? If the system can adjust to these changes, how long does it take for the system to adjust to changes? An ideal solution is a system that can easily and quickly adapt to changes in technology, laws, and the sUAS industry. After a change in technology, law, or the sUAS industry, an ideal solution would still be functional while possessing features that are relevant to managing sUAS traffic safely and effectively.
- **Adoptability**: Can the system be integrated into the current system for managing aircraft using the technology currently available? How much time, effort, and money do you need to effectively use the system? An ideal solution is a system that easily integrates into the current system for managing manned aircrafts and aircraft traffic while allowing various technologies to be used to interact and utilize the system. As a result, an ideal solution should require little time, effort, and money to use effectively.
- **Usability**: How much special knowledge or training is needed for a person to effectively use the system? How difficult would it be for an sUAS operator to use the system? An ideal solution is a system that would require little or no special knowledge or training, so basic or common knowledge would be sufficient to effectively use the system. However, the system should not impede a knowledgeable person from using the system easily and effectively. As a result, an ideal solution would make it easy for an sUAS operator, or any other human, to effectively use the system.
- **Performance**: How quickly can the system perform a particular function it was designed to perform? An ideal solution is a system that can quickly perform any of the tasks it was designed to perform in the manner it was specified to perform the task.

- **Reliability**: Will the system *always* do a requested task (if it was designed to do the task) within a finite period of time at any time? Will the system *always* perform the requested task in an expected and specified manner? An ideal solution would always be available and always perform any requested task it was designed to perform in the manner in which it was designed to perform it, at any time within a finite period of time.
- **Low-Cost**: How much money is needed to develop, deploy, and maintain the system? An ideal solution is a system can be sufficiently developed, deployed, and maintained without a large budget. It utilizes low-cost or free resources to function.

Chapter 2: The Proposed Solution

The proposed solution is a small UAS traffic management (sUTM) system that allows a sUAS operator to register himself with the system. To register with the sUTM system, the sUAS operator must provide some of his personal information, such as his name and email address. Having the sUAS operator register himself allows the system to contact him whenever necessary.

After the sUAS operator registers himself, he will register each sUAS he owns or will operate. Registering an sUAS allows the operator, or the sUAS itself, to report the location of the sUAS location in real time. This real-time reporting of sUAS locations allows sUAS operators to be aware of nearby sUAS and avoid these sUAS. Reporting the location of the sUAS also helps the operator find the sUAS if he loses sight of the sUAS and cannot control or communicate with it.

In addition to reporting the location of his registered sUAS in real time, the operator can create and submit a proposal to the system that includes when and where he will fly his registered sUAS. This proposal, called a *flight plan*, is then used by the system to authorize the sUAS flight.

To implement these features, the proposed solution consists of two major parts: a server and an Android mobile app communicating with the server.

2.1 System Architecture

2.1.1 Server

The server will be accessible through the Internet. As shown in Figure 4, the server communicates with the device with the Android mobile app installed. The server will receive, store, and manage the data of the registered sUAS operators and their registered sUAS. It will also receive and process flight plans. After it receives and processes a flight plan, the server will notify the sUAS operator (through the Android mobile app) that he has or does not have authorization to fly his sUAS in the area at the time specified in the flight plan.

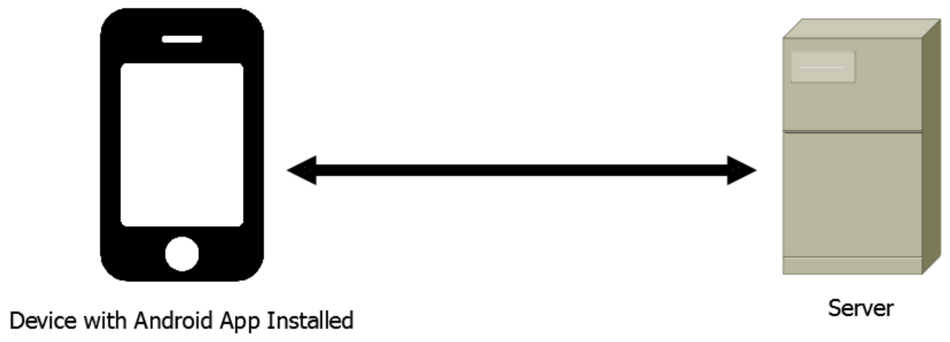


Figure 4 Architectural diagram of the sUTM system

Although the server will be accessible through the Internet, it will not act as a server for a website, where the user enters a URL and the server provides human-friendly interface. The server, when used directly, only provides data in the form of text that cannot be easily read by humans, but it can be processed by machines. This is because the server is intended to only communicate with the Android mobile app.

2.1.2 Android Mobile App

The server does not have an interface that allows a human to easily interact and communicate with it. The purpose of the Android mobile app is to serve as an easy-to-use interface to the small UAS Traffic Management (sUTM) functions provided by the server. Operators of sUAS will generally interact and communicate with the server through the mobile app. The app sends data provided by the sUAS operator to the server.

The mobile app is software that will be created to be used through the Android operating system. Android is one to the most widely used operating systems for mobile devices, such as smart phones and tablets. There are millions of mobile devices that use Android as their operating system. Creating an Android mobile app would allow millions of people to use this sUAS traffic management system without buying extra equipment. They could just use the smart phone they use every day to use the sUTM system. Additionally, developing, testing and deploying an Android app do not require any extra equipment or software that costs money. We can use a personal computer with average specifications to develop the app. After developing the app, we can deploy the app to the Google Play Store, where it will be easily available to the millions of people who have an Android mobile device.

2.2 Strengths of the Proposed Solution

- **It is somewhat adoptable**: Much of the proposed solution's adoptability comes from the fact that the mobile app can be used by millions of mobile devices that use the Android operating system. This allows many people to connect and use the sUTM system by using the smart phone they use every day and carry wherever they go. However, this adoptability is limited.
- **It is usable**: If the mobile app interface is designed correctly, a person can easily use the sUTM system without knowing exactly how the system works and have little knowledge of aviation rules and practices.
- **The amount of money needed for development is low**: Creating an Android app does not require any special equipment. The software needed to create an Android app is available to the public for free. Additionally, the server can be made sufficiently available on the Internet by purchasing an inexpensive domain name and an inexpensive web hosting package.

- **It can be reliable**: Because the server is not a human, it can run 24 hours a day, 7 days a week, for every day of the year (including holidays). This means the server should be able to perform any function it was designed to do at any time. However, the availability of the server depends on the hardware the server uses and how the software for the server is programmed. If the server uses faulty hardware or if the server's software malfunctions, the server may crash and not be available or give an unexpected result.

2.3 Required Features That Cannot Be Evaluated Yet

- **Maintainability**: Much of the maintainability of a solution depends on how the solution is designed, documented, and implemented. Because we have not discussed the design and implementation of the proposed solution yet, we will not discuss maintainability of the proposed system in this chapter.

- **Performance**: Similar to maintainability, the performance of a solution depends on how the solution is design and implemented. But it also requires the system to be tested for performance in various ways. Therefore, we are not going to evaluate the performance of the proposed solution in this chapter.

2.4 Shortcomings of the Proposed Solution

- **Adaptability is severely limited**: The proposed solution relies on an Android mobile app to interact with the server. What if a person has an Apple iOS or Windows device? What if a person needs to use a personal computer? What if the Android operating system is not popular anymore? Because the only easy way for a person to use the server is to use the Android mobile app, a person who does not have an Android mobile device has little or no hope of using the sUTM system. A person who wants to use a different type of device would have to wait until we create software for that device.
- **Adoptability is limited**: Not everyone can or is willing to solely use an Android app to interact with the sUTM system. By forcing people to use the Android mobile app we develop, we exclude many sUAS operators from the sUTM system. This is not what we want. We want to include as many sUAS operators into the system as possible.

2.5. Revising the Proposed Solution to Overcome Its Shortcomings

As discussed in the in section 2.4, the sUAS traffic management (sUTM) system in the proposed solution has limited adaptability and adoptability. Fortunately, these shortcomings can be nearly eliminated by making the server host a *public* web service. A web service is a software system designed to support interoperable machine-to-machine interaction over a network. [6] In the case of sUTM system in this proposed solution, the network would be the Internet and the machine-to-machine interaction would be the interaction between server and whatever device wants to communicate with the server. If the web service is public, the server's functions and services will be available to any software or Internet-enabled device. The server’s functions are available through a well-documented and publicly available application programming interface (API). An API is a set of rules and functions that govern how one application is able to communicate with another application. [7] A well-documented API enables any software developer to easily integrate the server's sUTM functions into their own software. In fact, it is highly encouraged that software developers develop software utilizing the web service, as it helps increase sUAS safety through increased usage of the sUTM system. By allowing any software or Internet-enabled device to use the server, the sUTM system can easily adapt to changes in technology, as long as the new technologies can use the Internet.

Along with these changes, we will still develop an Android mobile app. However, the mobile app will serve primarily as a functional example of software utilizing the server. Providing a functional example is useful for software developers to understand how to use the server's API. Because the Android mobile app will serve primarily as an example, it does not need to try to satisfy every possible use case. However, the quality of the app still matters. Because the server does not have a human-friendly interface, the Android mobile app we create will eventually function as a marketing material that can attract sUAS operators and software developers to use the sUTM system. The app would also provide the first impression of how well the system works for sUAS operators or software developers. A positive first impression can compel sUAS operators and software developers to frequently use the sUTM system.

2.5.1 Difference between This New Version of the Proposed Solution and the Previous One

With this new version of the proposed solution, there is still a server and an Android mobile app similar to the previously proposed solution. The server in this solution performs the same functions as the server in the previous solution. The server in the previous version of the solution would also need a web service and an API to communicate with mobile app. Then, what is the difference between this solution and the previous solution?

The biggest difference lies in the change in the importance and the purposes of the server and the Android mobile app. In the previous version of the solution, the importance of the roles of the server and the mobile app were equal, as the system would not be useful without one or the other. On the other hand, in this new version of the solution, the server has a more vital role than the mobile app because the system does not rely on the mobile app we develop to function effectively. This is because of the substantial change the in the sUTM system's architectural structure, shown in Figure 5. As mentioned before, the Android mobile app we develop would serves primarily as a functional example of software utilizing the server. Because the server's functions are now more available and easier for software developers to integrate into their own software, another software developer can easily develop an Android mobile app if we fail to do so. A user can just use some other available software if our mobile app fails or is not suitable for their needs. On the other hand, if the server fails, any software or device that relies on communicating with the server to function would also fail. This fact increases the need for the server to be reliable and always available.

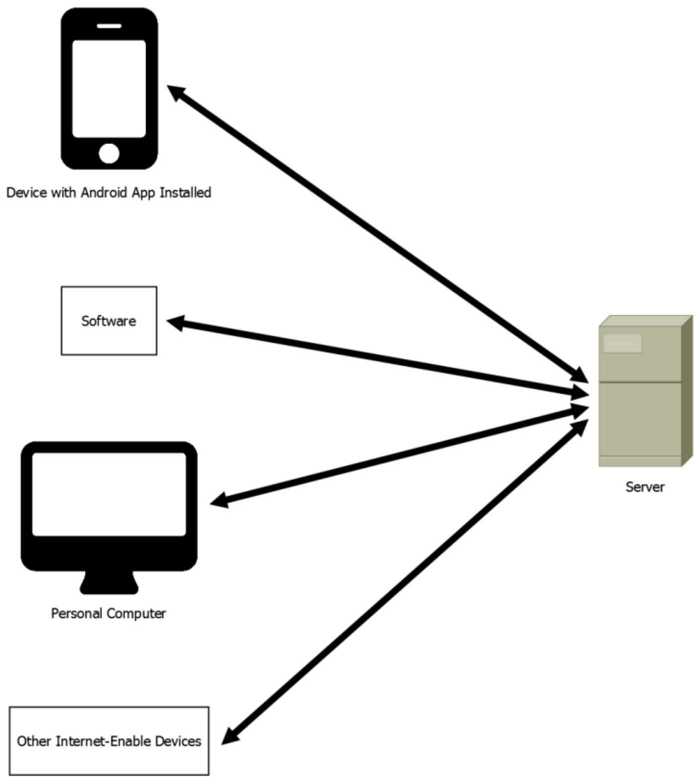

Device with Android App Installed
Software
Personal Computer
Other Internet-Enable Devices
Server

Figure 5 Architectural diagram of the sUTM system where the server is hosting a web service

Chapter 3: Server Design and Implementation

3.1 REST Software Architectural Style

The application programming interface (API) for the web service hosted on the server needs to be designed in a way that allows software developers to quickly learn and use the API. To fulfill this requirement, the web API should use the REpresentational State Transfer (REST) software architectural style. REST is used in many other widely-used web APIs including the web APIs for Twitter [8] and Google Maps [9]. This will make our web API easier to learn for the many software developers who are familiar with the more popular web APIs. Note that REST is only a software architectural style, so it does not enforce a specific implementation or protocol. [10] However, our web API will use the Hypertext Transfer Protocol (HTTP) to implement REST. This is because the web API we want the sUAS traffic management (sUTM) system to utilize the Internet, and HTTP the most commonly-used protocol for accessing the Internet.

In the REST software architectural style, there is a *server* and at least one *client*. [10] The server is the machine or system that provides particular services for other devices or software. The client is a device or software that communicates with the server. According to this new terminology, the Android mobile app we develop is a *client* and the server we create is obviously the *server*. Generally, there is one server and multiple clients communicating with the server at different times or possibly at the same time. No coordination between the clients is needed, so a client does not need to know the application states of the other clients, or even if other clients exist. This characteristic of the REST software architectural style allows multiple clients to communicate with the server at any time.

Generally, a client communicates with the server to get some data from the server or to ask the server to perform some task. In order for to server to know what task the client wants it to perform, the client must provide the name of the operation it wants the server to perform and some identifier of the object (a set of data) on which the client wants the specified operation performed. An operation, such as "create" or "delete", is performed on an object, which is called a *resource*. Each resource that is provided by the server is identified by a unique ID, which is used in the Uniform Resource Identifier (URI) for the resource. A URI is similar to a Uniform Resource Locator (URL), but it does not indicate the location of the resource.

In this HTTP implementation of REST, the operations the server is able to perform are the HTTP methods, which include "GET", "POST", "PUT", "PATCH", "HEAD", and "DELETE". The client requests the server to perform some task by sending the server a combination of one of the HTTP methods and the URI of a resource. These HTTP method and URI combinations make up much of the REST API. In addition to a HTTP method and URI combination, the client may also send additional data the server needs in order to perform the task requested by the client. Table 1 is a list of some of the HTTP method and URI combinations in the API. The full list of the HTTP method and URI combinations is contained in the appendix. In Table 1 and the appendix, the part of the URI that are between the curly braces ("{" and "}") are names for the parts of the URI that must be provided by the client, such as a unique ID for the resource. In other words, the curly braces act as placeholders for actual values in the URIs. For example, the "/users/{username}" URI becomes "/users/johnDoe" if "johnDoe" is the unique username of the user.

After the server receives a request from a client, it decides if it should or is able to perform the task. The server then gives the client a response, even if it refuses or is unable to perform the task requested by the client. When the client receives the server's response, the client processes the response and decides what to do with the data in the server's response. When a server sends a response to a client, its response has a three-digit code indicating the type of response. These three-digit codes are part of the HTTP specification (RFC 2616) and are called *HTTP status codes*, or simply *status codes*. [11] A HTTP status code ranges from 100 to 599. Table 2 is a list of a few of the common status codes and their meaning.

HTTP Method	**URI**	**Task to be performed by the server**
GET	/users/{username}	Get the information of the user with the given username
POST	/users	Register a new user by creating a new user account
PATCH	/users/{username}	Update the information of the user with the given username
GET	/uas	Get all UAS that match the given criteria
POST	/uas	Register a UAS by creating a new resource for the UAS
GET	/uas/{uasId}	Get the information of the UAS with the given ID (“uasId”)
DELETE	/uas/{uasId}	Delete the UAS with the given UAS ID
GET	/flight-plans	Get all flight plans that match the given criteria
POST	/flight-plans	Create a flight plan for a registered UAS
GET	/flight-plans/{flightPlanId}	Get the information of the flight plan with the given flight plan ID (“flightPlanId”)
PUT	/flight-plans/{flightPlanId}	Update the information of the flight plan with the given flight plan ID

Table 1 Some of the HTTP Method and URI combinations available within the REST API

HTTP Status Code	**Meaning**
200	**OK -** The requested task was successfully completed by the server.
404	**Not Found -** A resource with the given URI could not be found. There could be a typo in the URI, or the resource that had the URI was deleted.
503	**Service Unavailable -** The server is currently unavailable. The server is usually unavailable because it is undergoing maintenance, but it could also be unavailable because it was overloaded.
500	**Internal Error -** The server encountered some unexpected error while processing the request. This error is typically given if the person or the people who created the server's software made a mistake somewhere in the server's software.

Table 2 A list of a few of the common HTTP status codes

3.1.1 Example REST Communication

Suppose a client wants to register a user. The client sends “POST /users” to the server along with the some information about the user that the server will store. In this case, the HTTP method “POST” is the operation to be performed on the resource with “/users”, so “POST /users” can be translated as, “Create (Register) a new user and add (post) it to the users list”. After receiving this request, the server creates a new user with information given by the client. The server then responds to the client with a status code of 200 and gives the data of the newly registered user. This response can be translated as the server saying, “Everything was OK on my end. I successfully created (posted) the new user, here's current information that I have about the new user.” Figure 6 is a diagram that demonstrates this example interaction.

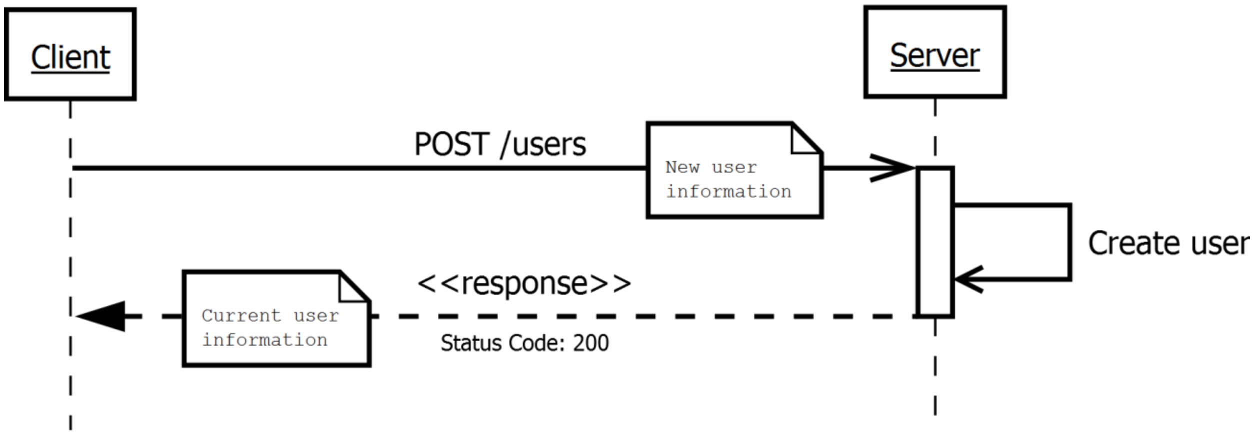


Figure 6 Diagram of an example REST interaction between a client and the server

3.2 User Authentication (“Logging In”)

For security purposes, some of the functions of the server require the sUAS operator to authenticate herself (or “sign in”), by using a client, to get the server to perform the requested task specified. In this paper, we will name the process of the sUAS operator authenticating herself (through a client) *user authentication*. As indicated in the title of this section and mentioned in the previous sentence, user authentication is better known as the “logging in” or “signing in”. Also, the word *user* in this context would refer to the human using a client. The user and the client are *not* the same. To summarize this terminology: a *user* (the human sUAS operator) uses a *client* (e.g. an Android mobile app) to interact with the *server*.

Surprisingly, there is currently no widely-accepted protocol or method for user authentication in RESTful web service APIs (APIs for web services that use REST). This is because creating a stateless method for user authentication is difficult.

3.2.1 Requirements for User Authentication for the Web Service

The user authentication scheme for the web service must:

- Prevent a client not authorized by the user from acting on the user's behalf.
- Be easy for software developers to use effectively and securely.
- Require a low number of transfers between the client and the server.
- Require only a small amount of data to be sent between the client and the server.
- Not rely heavily on the Secure Sockets Layer (SSL). SSL should be used just as an extra security layer whenever possible. In other words, SSL should be used to increase the *defense-in-depth.* [12]
- Be *stateless*. Being stateless means that the server does not have to maintain a list of sessions, tokens, or other data in its database. This means the client provides all of the information the server needs to authenticate as the user.

3.2.2 Most Popular Solutions for Rest API User Authentication

3.2.2.1 Traditional (Cookie-Based) Authentication

This is the method of user authentication used by a typical website that serves web pages. To authenticate (“log in”) using the traditional authentication method, the user provides her username and password to the server. When the server receives the user’s username and password, it creates a *session token*, which is a randomly generated string of characters (letters, numbers, and symbols) used by the server to check if the user has authenticated. A server using traditional authentication usually stores the session token in its database. After creating and storing the session token, the server creates a *cookie*, which is a set of data that contains the session token. The server then sends the cookie to the web browser, which is a client in this scenario. The web browser stores the cookie in a temporary file on the user's computer.

One of the biggest problems with the traditional authentication method is that the session token for each authenticated user is stored in the database. This is considered to be *stateful* (the opposite of *stateless*, which was explained in section 3.2.1), as the server stores the application state of the web browser. This *statefulness* causes the server to use the database more frequently when a user logs in. The extra database usage creates more work for the server. As a result, the system becomes less manageable as the number of users increase.

Another one of the biggest problems with traditional authentication is that it depends on the use of cookies. Because support for cookies is feature typically only found in web browsers, it is difficult for a client that is not a web browser, such as a mobile app, to authenticate as the user.

3.2.2.2 OAuth

OAuth is currently the most popular user authentication solution for RESTful web services. OAuth, as described by its official website, is "[a]n open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications." [13] The latest version of OAuth is OAuth 2.0. The widespread adoption of OAuth 2.0 seems to be a result of the limited number of options for user authentication for RESTful web services. Interestingly, OAuth 2.0 is technically not a protocol. As stated in its formal specification, RFC 6749, OAuth is a "framework". [14] OAuth 2.0 is not a protocol because it allows too many variations. Because of the many variations, two implementations of OAuth 2.0 could be vastly different and unable to communication with each other.

Because OAuth is complex and was originally designed for web browsers, it is difficult to implement OAuth 2.0 securely for a mobile client (mobile app) without some expertise in web security. [15] Additionally, OAuth 2.0 relies heavily on Secure Sockets Layer (SSL) for security. If an SSL certificate error is ignored, the whole authentication scheme becomes useless as a security measure. [16]

3.2.2.3 HTTP Basic with SSL

Using HTTP Basic for user authentication is only secure when it is used with SSL because HTTP Basic sends the user's credentials (username and password) in plain (unencrypted) text. However, solely relying on SSL for security is very limited. If SSL fails for some reason, the user's credentials are exposed. Also, if the client, for some reason, does not send an HTTP request containing HTTP Basic credentials over SSL, that request was sent with the credentials in plain text. As a result, the only security measure was compromised, even if the server did not accept the request. [17] Because of the problems with HTTP Basic over SSL, it should not be used to authenticate the user in every HTTP request. However, HTTP Basic with SSL is usually sufficiently secure enough to be used as an easy way to occasionally send user's credentials to the server in order to create and obtain a set of temporary credentials (e.g. set of random or cryptographic strings) that can then be used to authenticate the user in every request afterwards.

3.2.2.4 HTTP Digest

HTTP Digest is considered to be better than HTTP Basic because the user's credentials (username and password) are encrypted before they are sent. However, HTTP Digest encrypts credentials using the MD5 hashing algorithm, which is an algorithm vulnerable to brute force attacks. Because of the use of the MD5 algorithm for encrypting the user’s credentials, HTTP Digest is also vulnerable to brute force attacks, where a hacker repeatedly tries every possible password until he finally gets the correct one. [18] Additionally, HTTP Digest requires the client to negotiate with the server for computing the nonce (<u>**n**</u>umber used <u>**once**</u>). [19] This negotiation requires an increased number of transfers between the client and the server for a single client request. This increase number of transfers is costly for clients that use certain mobile data networks (3G, 4G, 4G LTE, etc.) for accessing the Internet because it costs money for each transfer. Additionally, the increased number of transfers is costly for the server because the extra connections consume the server's resources, such as connection bandwidth and memory.

3.2.3 The sUTM System's User Authentication Scheme

For the system's user authentication scheme, we will create and use a variation of the Oz protocol. Oz is a protocol designed to be a simple-to-use and secure user authentication solution that is based on the OAuth 1.0a protocol. [20] In fact, Oz was created by Eran Hammer, one of the authors and main contributors of the OAuth 1.0 and OAuth 1.0a specifications.

In our variation of the Oz protocol, we will remove the requirement for the client to register with the system and authenticate itself with the server. This requirement will be removed because we currently do not need client authentication and the client authentication portion of the protocol adds complexity that is currently unnecessary. However, client authentication will eventually become necessary to enforce certain client restriction, such as usage quotas. Since we are currently not enforcing any server usage restrictions, we not require client authentication for now.

3.2.3.1 User Authentication Process

Instead of the client sending the user's username and password for every request to the server that requires authentication, the user's credentials (username and password) is sent once to get a new set of temporary credentials, which is a set of random and cryptographic strings. These temporary credentials are stored within the client in the form of a *user authentication ticket*. A user authentication ticket (or simply *ticket*) also includes other information a client needs to authenticate as the user. After the client receives a ticket from the server, it uses the temporary credentials stored within the ticket to authenticate as the user in subsequent API requests. By using the temporary credentials, the user's credentials are not sent in subsequent requests. This feature reduces the risk of exposing the user's credentials to a hacker. It also allows a ticket to be revoked without the user needing the reset or change her password.

As shown in Figure 7 and Figure 8, the normal user authentication flow is as follows:

1. The client sends the user's username and password as an HTTP request by using HTTP Basic authentication with SSL. The server will reject the client's request and return an error message if the client sends the user's username and password without SSL.
2. The server checks if the username and password combination is correct.
3. If the username and password combination is correct, the server responds to the client by issuing a ticket to the client. The ticket contains the ticket ID (public, allowed to be known by anyone) and the ticket secret key (not public, only client and server should know secret key). The ticket's ID and secret key are generated by the server. The ticket's ID is a cryptographic string of characters, not the user's username, so that the user can authenticate requests using different clients (apps or devices) at the same time. This enables each client the user uses to have a different set of credentials (ticket ID and secret key) that is separate from the other clients. The ticket's secret key is a randomly generated string, not the user's password. If the ticket's secret key is somehow exposed to something or someone other than the server or the client, the ticket can be revoked and made invalid without the user needing to change his/her password.
4. Now the client can use the ID and secret key of the ticket to authenticate as the user in subsequent requests.

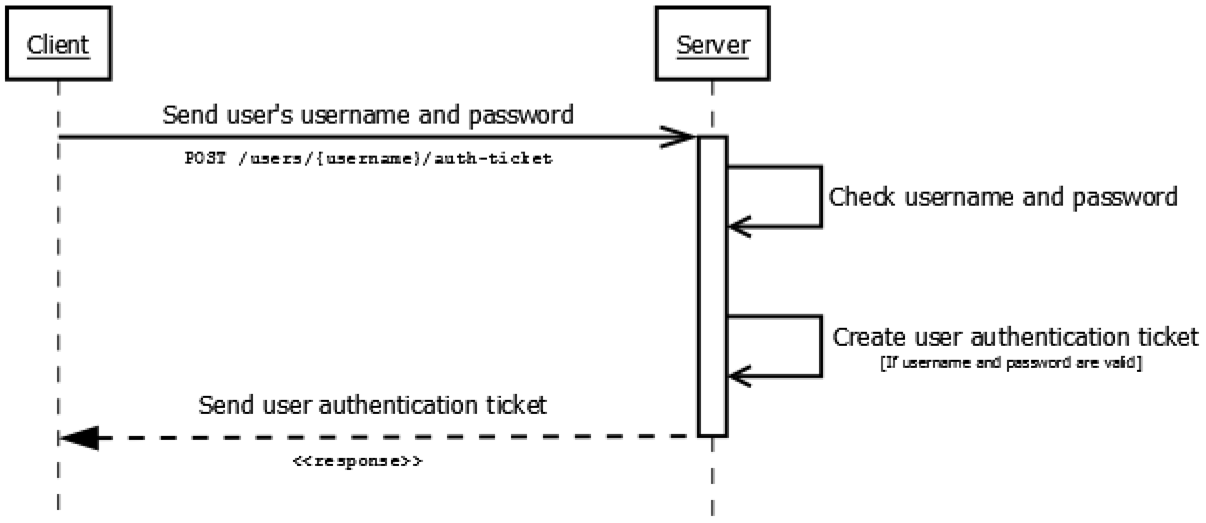

Client
Server
Send user's username and password
POST /users/{username}/auth-ticket
Check username and password
Create user authentication ticket
[If username and password are valid]
Send user authentication ticket
<<response>>

Figure 7 Diagram of the process for the client to obtain a ticket for user authentication

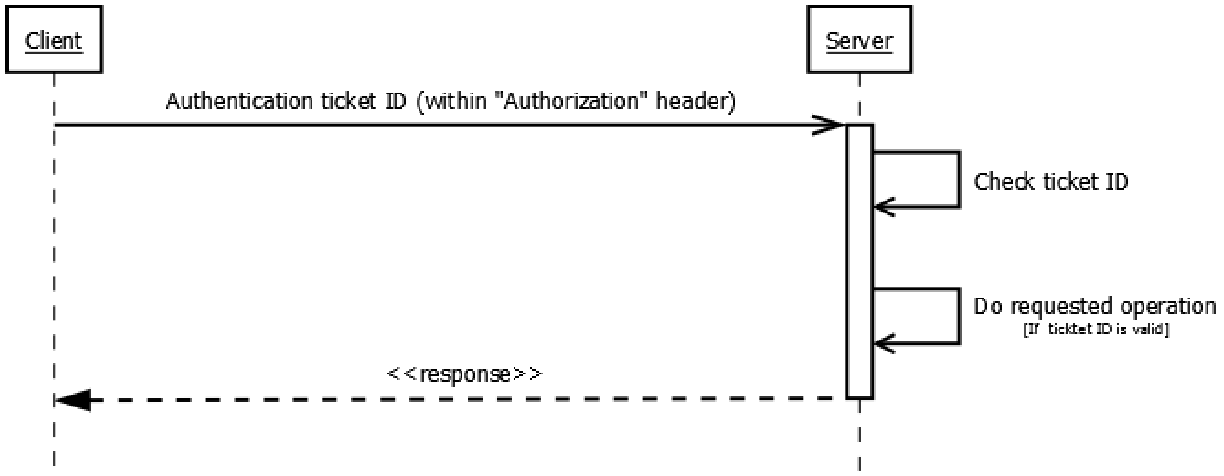


Figure 8 Diagram of the process for the client to authenticate as the user after obtain a ticket

3.2.3.2 “Logging Out”

In the commonly-used traditional (cookie-based) authentication, a user can “log out” by commanding the client to request the server to destroy the user's current session the server is storing.

After the server destroys the user's current session, no one can authenticate as the user through the client without providing the user's credentials (username and password) again.

With our REST API user authentication scheme, there is no need for the client to request to the server to destroy the user's current session, because the server never stores a user session. If a user wants to “log out” to prevent anyone from authenticating as her through the client, the client only needs to destroy the authentication ticket it was storing and using to authenticate as the user. To authenticate as the user again, the client must repeat the process shown in Figure 7 to get another authentication ticket.

5.2.3.3 Ticket Expiration

In addition to the temporary credentials, a user authentication ticket contains the exact date and time the temporary credentials (ticket ID and secret key) will expire. The temporary credentials stored in the ticket expire 60 days after the ticket was created and issued by the server. An expired set of temporary credentials stored in the ticket cannot be used to authenticate as the user. After the ticket credentials expire, the client must ask the user for her username and password to retrieve another ticket in order to authenticate as the user again.

3.2.3.4 Revoking Tickets

If a ticket's secret key is exposed to someone or something other than the client and the server, the client can ask the server to revoke the ticket. When the server revokes a ticket, the temporary credentials (ticket ID and secret key) stored in the ticket are invalidated and not accepted by the server, even if the ticket has not expired yet. To revoke a ticket, the server adds the ticket ID to a blacklist stored in database. When the server receives a ticket, the server must check if the temporary credentials it received are from a ticket has been revoked. Unfortunately, this adds a little bit of statefulness to the user authentication scheme because the server maintains the state of a revoked user authentication ticket. However, this statefulness is (or should be) small enough to be manageable as the system increases in size because the need to revoke a ticket should be infrequent. With the trade-off between statelessness and security, the increased security at the cost of a minor loss in statelessness is worth the price.

3.2.3.5 Server Storage of Tickets

The server usually does not need to store tickets because it can get all the information it needs by decrypting the ticket ID the client sends with the secret encryption password. This feature allows the sUTM web service to remain mostly stateless. As mentioned in Section 3.2.3.3, this authentication scheme is not entirely stateless because of the one exception where the server must maintain a blacklist (a type of state) of revoked tickets and check if each ticket used in a request has been revoked (blacklisted).

3.2.3.6 Role of a Ticket's Secret Key

Because a ticket's secret key is randomly generated by the server, each secret key should be unique to that ticket, but it is possible (but very improbable) for two tickets to have the same secret key. The primary purpose of a ticket's secret key is to allow the server to confirm that the ticket ID included in a request is the ID of the same ticket that was issued to that client. Because only the server issued the ticket to the client, the server has the ticket's secret key. Using a secret key prevents an attacker from being able to use a stolen ticket ID to authenticate as the user in a request to the server because the attacker needs both the secret key (never sent in unencrypted, plain text in a request) and the ticket ID to successfully authenticate as the user.

3.2.3.7 Client Storage of the Ticket Secret Key

The ticket's secret key must be stored in plain text. Because the client does not need to send the server the secret key it received, the storage of the secret key is an issue for clients. For a client that is a web browser, the secret key can be safely stored in plain text within local storage or session storage. [21] [22] [23] Storing the secret key within a cookie should be avoided. [22] For a client that is not a web browser (e.g. a mobile app), the secret key can be stored in a file or in memory.

3.3 Flight Plan Authorization

After a registered sUAS operator registers his sUAS, he can create a flight plan for that sUAS whenever he plans to operate the sUAS. After sUAS operator creates a flight plan, he submits the flight plan to the server in order to obtain authorization for the sUAS flight specified in that flight plan. Afterwards, the server processes the submitted flight plan and determines if the sUAS flight proposed in the flight plan should be given authorization. When a flight plan is given authorization, the system has determined that the sUAS flight proposed in the flight plan is safe enough. The server's process for determining if a flight plan should be granted authorization will generally be automated. The server's decision of whether a flight plan should be given authorization is determined by the properties of the sUAS, the time of flight, the purpose of the flight, and a few other factors. The server's rules for deciding whether or not a flight plan should be granted authorization is based on current U.S. federal laws, guidelines, and regulations regarding the use of small UAS and aircraft in general.

Sometimes, a flight plan's authorization depends on whether or not air traffic control (ATC) authorizes the flight. An example of this instance is when the sUAS flight is for a recreational or hobby purpose and the sUAS will be operated within 5 miles of an airport. [24] In such a case, authorization is needed from the ATC that has jurisdiction over that nearby airport. [24] The server will not grant flight plan authorization until ATC authorizes the flight. If ATC rejects authorization for the flight plan, the server will also reject authorization for that flight plan. Through the use of the REST API, the server can notify ATC whenever a flight plan needs their authorization. ATC can then grant or reject authorization of that flight plan by using the REST API. This allows ATC to be notified of only the flight plans that needs their authorization. ATC does not need know about the flight plans that do not need their authorization. This means that ATC is notified only whenever they need or want to be notified. This makes sure that ATC is not overwhelmed with notifications for sUAS flight plans that do not need their attention, which allows ATC to continue to manage manned flights while managing sUAS flights.

3.3.1 Current Limits of the Server's Automated Flight Plan Authorization System

- **No commercial sUAS use**: The server's automated flight plan authorization system does not have any rules regarding any of the currently legal commercial uses of sUAS. As a result, the server will not grant authorization for any flight plan for any commercial use of an sUAS.
- **sUAS size limit**: The automated flight plan authorization system does not use any rules regarding the legal use of large UAS (UAS that weigh more than 55 lbs.) [25], so a flight plan for a large UAS will not be granted authorization. This limitation is intentional, as this system is designed for small UAS.
- **Temporary Flight Restrictions (TFR) Areas**: The automated authorization system does not account for the TFR areas where sUAS flight would not be allowed. This is because machine-readable data about the TFR areas cannot be easily obtained.
- **Local laws and regulations**: State and municipal (city) laws can also affect whether an sUAS is allowed to fly. Because there are 50 states and thousands of cities in the U.S., the server does not have any rules based on state or municipal laws. Attempting to create such rules would be too time consuming and difficult to maintain.

3.3.2 Future Development of the Server's Automated Flight Plan Authorization System

The server's automated flight plan authorization system is incomplete. The authorization system is not very useful in its current state. More rules for determining flight plan authorization need to be extracted from the U.S. federal laws, guidelines, and regulations. Additionally, some sets of aviation data the server is currently unable to obtain and process in an automated manner. This is a problem because some of the rules in the authorization system depend on these data sets. For some sets of data, there is a way to obtain the data in an automated manner, but there is no easy way for the server to process the data. This is because the data is in a format that is designed to be read and understood by humans, but is not easily read and processed by a machine.

Chapter 4: Android Mobile App Design and Implementation

Because the Android mobile app serves primarily as an interface for the server, most of the design of the app is the user interface (UI), which is the interface the user (sUAS operator) will interact with. The primary concern about designing the user interface is user experience (UX). There are many components of UX, including the navigation structure, appearance, and content.

4.1 Navigation Structure

The navigation structure of a UI refers to the perceived structure of interconnected sub-interfaces. A user often uses the navigation structure to figure out what she is able to do at any moment while using the UI. The design of the navigation structure and the mechanisms for interacting with the navigation structure is referred to as *navigation design*. [26] If the navigation structure of a UI is designed correctly, a user can use the UI with little effort.

For this mobile app, the navigation structure will be similar to that of a web site. However, we will not refer to the sub-interfaces as *pages*, as you would with a web site. Instead, we will refer to these sub-interfaces as *views*. As shown in Figure 9, the user always starts at the “Home” view. From the “Home” view the user has various options of where to go next. In general, each view has a singular purpose. This eliminates the information that is unnecessary for performing any of the tasks currently available to the user. This allows the app to maximize the usage of the limited amount of screen space while allowing the user to process only the information that is relevant to the task she wants or needs to perform.

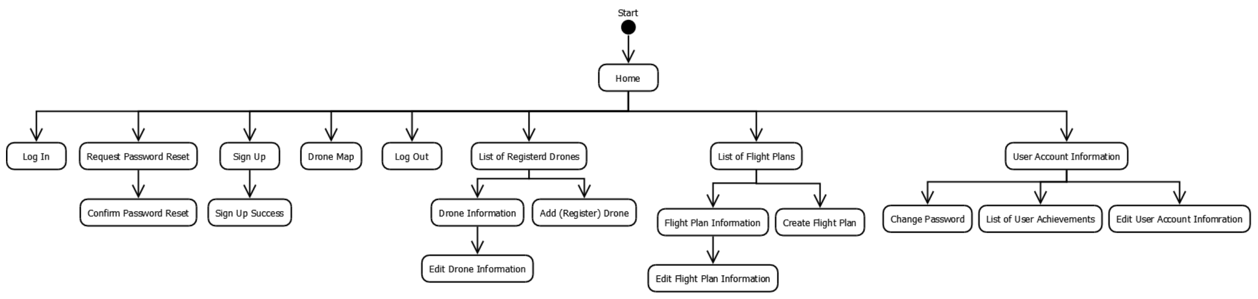


Figure 9 Diagram of the navigation structure of the Android mobile app

4.2 Appearance

As mentioned in Chapter 2, this Android mobile app will typically be the first impression an sUAS operator or software developer will have of the system. We want this first impression to be positive. To make this positive first impression the app needs to look attractive (at least to most people). However, the appearance of the app must not only be attractive, but functional as well.

4.2.1 Font

The app UI can use a serif font or a sans-serif font. A serif font is a font where the each letter has a small line, called a *serif*, at its edges. On the other hand, a sans-serif font is a font where each letter does not have serifs. Figure 10, Figure 11, and Figure 12 are examples that demonstrate the difference between a serif and a sans-serif font.

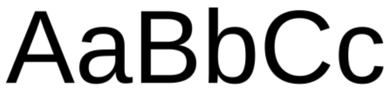


Figure 10 An example of a sans-serif font. Image Source: [27]

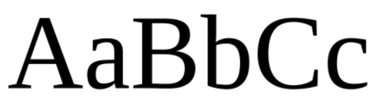


Figure 11 An example of a serif font. Image Source: [28]

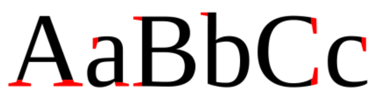


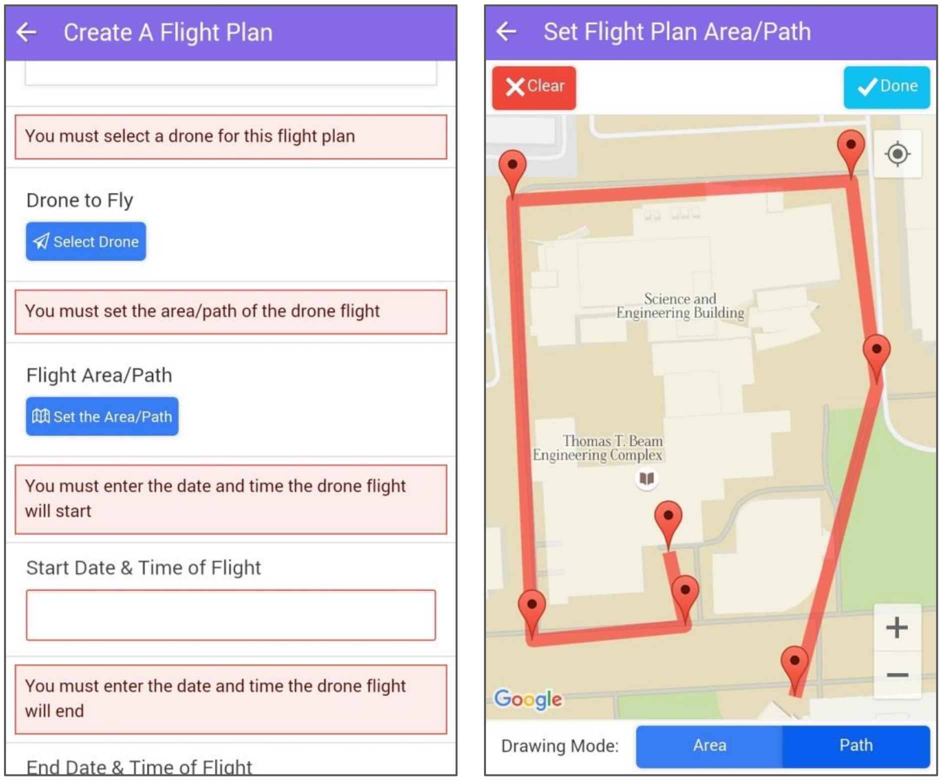
Figure 12 The serif font in Figure 11 with the serifs highlighted in red. Image Source: [29]

Because the Android mobile app will often be installed on a device with a small screen, UI needs to be as readable as possible on a small screen. In general, sans-serif fonts are more readable on smaller screens than serif fonts because the font size is also smaller. [30] This is why the UI uses a sans-serif font instead of a serif font.

4.2.2 Color

What colors should we use for the UI of the app? We should use only two different colors, other than white, black, and gray, for the UI's primary color scheme because using too many colors can make the UI look overly complex. To help guide our decision of which colors to choose, we will use color psychology. Color psychology is the study of how color influences psychological functioning. [31] According to the *Handbook of Disease Burdens and Quality of Life Measures*, psychological functioning is an individual’s ability to his/her goals within him/herself and the external environment. [32] Psychological functioning includes an individual’s behavior and emotion. [32] Despite the fact that color psychology is an inexact science that is backed by an insufficient amount of scientific data, it is often used in marketing to influence a consumer's behavior.

There are two types of colors: warm colors and cool colors. Warm colors are colors such as red, yellow, and orange. Cool colors are colors such as blue, green, and purple. The use of warm colors often increases impulsive actions while the use of cool colors often decreases impulsive actions. [33] In our case, we do not want an sUAS operator to operate an sUAS by using their impulses. An sUAS operator operating a sUAS according to his impulses could endanger manned aircraft, nearby people, and the operator himself. Instead, we want the sUAS to slow down and think rationally about what he is doing. As such, using cool colors as the primary color scheme in the UI of the app is the best option. However, we can still use the warm colors make certain messages or other UI elements, such as error messages and alerts, stand out from the rest of the UI. The use of warm colors in a few views is shown in Figure 13.


Create A Flight Plan
Drone to Fly
Select Drone
Flight Area/Path
Set the Area/Path
Start Date & Time of Flight
End Date & Time of Flight
Set Flight Plan Area/Path
Clear
Done
Science and Engineering Building
Thomas T. Beam Engineering Complex
Google
Drawing Mode:
Area
Path

You must select a drone for this flight plan

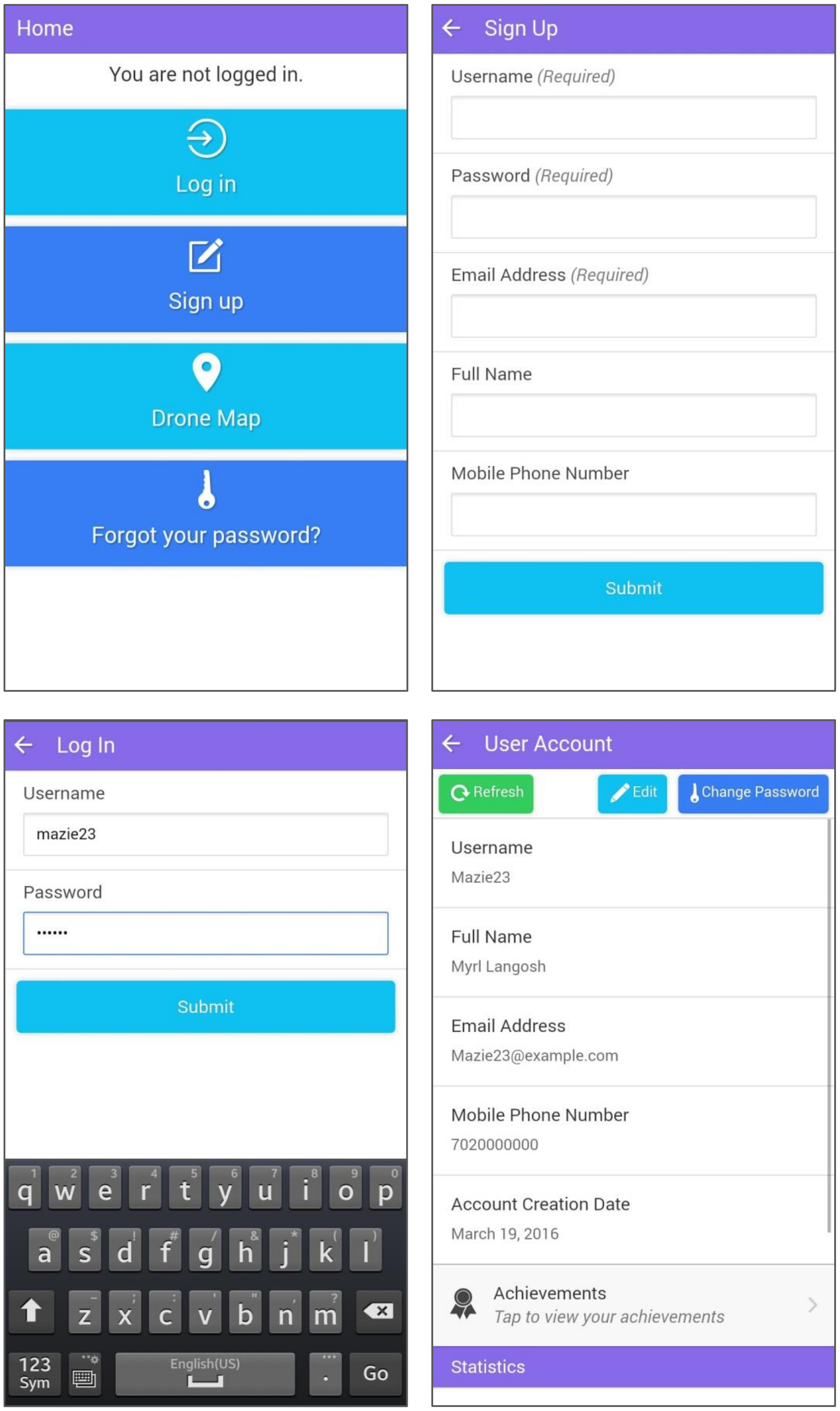
You must set the area/path of the drone flight

You must enter the date and time the drone flight will start

You must enter the date and time the drone flight will end

Figure 13 Screenshots of views that use red, a warm color, to make certain messages and UI elements stand out

By choosing to use cool colors in our primary color scheme, we reduced our options to three colors: blue, green, and purple. We should use blue because sUAS usually fly in the sky and the sky is normally blue during the daytime. Now we must choose between green and purple. We will choose purple instead of green because purple exhibits a sense of authenticity and quality. [34] We want the user of the app to think that the app and the system are well-designed and well-made. However, we can also use green to highlight certain elements in the UI. The screenshots in Figure 14 demonstrate how the chosen UI color scheme is used throughout various views in the app.



Username
mazie23
Password
••••••

Username
Mazie23

Full Name
Myrl Langosh

Email Address
Mazie23@example.com

Mobile Phone Number
7020000000

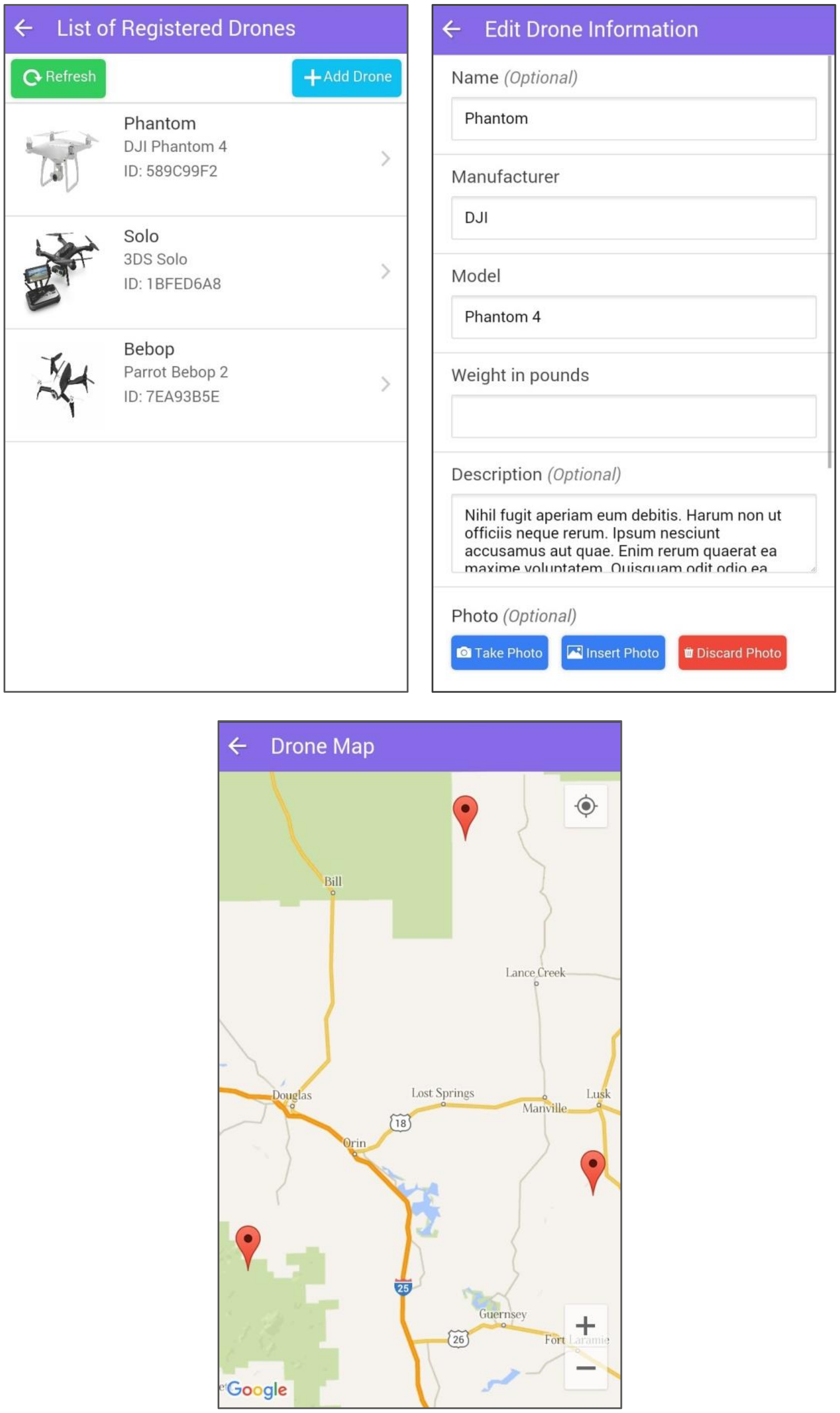
Account Creation Date
March 19, 2016

Figure 14 Screenshots of views using the chosen color scheme

4.3 Content

Whenever possible, the text in the app UI does not use jargon. Using jargon in the app UI text makes it more difficult for an inexperienced sUAS operator to use the app. Additionally, the vocabulary used in the text is the vocabulary that would be used by a person who has little to no UAS or aviation knowledge, even if certain words and word usages that are not preferred by the UAS industry.

For example, the app UI uses the term *drone* instead of *UAS* or *Unmanned Aircraft System*, which is demonstrated in the screenshots in Figure 15. Although the UAS industry prefers to use the standard terms *UAS* and *Unmanned Aircraft System* over the word *drone*, the word *drone* is more popular among those who are not involved the UAS industry. [35] In many cases, those who are not involved in the UAS industry may not know what a *UAS* or *Unmanned Aircraft System* is, but they do know what a *drone* is. [35] Additionally, the word *drone* is more catchy and identifiable because it is only one syllable. Because of these facts, it is better for the app UI to use the word *drone* instead of *UAS* or *Unmanned Aircraft System*.


List of Registered Drones

Refresh | Add Drone

Phantom
DJI Phantom 4
ID: 589C99F2

Solo
3DS Solo
ID: 1BFED6A8

Bebop
Parrot Bebop 2
ID: 7EA93B5E

Edit Drone Information

Name *(Optional)*
Phantom

Manufacturer
DJI

Model
Phantom 4

Weight in pounds

Description *(Optional)*
Nihil fugit aperiam eum debitis. Harum non ut officiis neque rerum. Ipsum nesciunt accusamus aut quae. Enim rerum quaerat ea maxime voluntatem. Quisquam odit odio ea

Photo *(Optional)*

Take Photo | Insert Photo | Discard Photo

Drone Map

Figure 15 Screenshots of views that use the word *drone* instead of *UAS*

Chapter 5: Testing Strategies

5.1 Unit Testing

Automated tests for the functionality of the server functions were created during the development of the server. These tests were designed to check if the program components for the server functions functioned according to how they were specified. Tests for checking if the components of a program function according to a specification are called *unit tests* because each component or *unit* of the program is tested. The activity of creating and running unit tests is referred to as *unit testing*. Most of the time, the unit testing process is automated. One reason for automated unit testing is that it makes software testing cheaper and more reliable while allowing software developers to save time testing the software. [36] Another reason is to make the server software more maintainable and reliable.

The automation of the unit tests can save a huge amount of time spent on testing. For example, if automated unit tests are created for all of the components of a medium-sized program, an entire program can be tested in less than an hour (or within a few minutes). Manually testing an entire medium-sized program would typically take at least a few hours (or at least an entire work-day, which is about 8 hours). In general, unit tests do not check if the design and specification of a program satisfies the needs of those who will use the system. The only way to test the design of the API is to extensively use the server through the API. Much of this type of testing is done by using the API when creating the Android mobile app. The unit tests do not test the performance of the server's software. This means the unit tests do not test how fast the sUTM server processes and responds to a client request and how much computing resources (memory, connection bandwidth, hard drive space, etc.) the server uses to process and respond to the client's request.

Unit testing during the initial phase of developing software creates more work during this phase and makes this initial software development process longer. This is why unit testing is an activity that is often delayed or neglected entirely in many other software projects. However, the cost of not unit testing the components of the program increases later in development. This cost is usually in the form of the amount of time and money spent to develop the software and fix problems with the software. The amount of increase in the cost of not unit testing and how quickly this cost increases depends on the nature of the software project. For some software projects, the cost grows exponentially as more of the software is developed. This cost is often the greatest and most prevalent when maintaining, extending, or optimizing the software. The cost of maintaining software with no unit tests eventually exceeds the cost of programming the functionality of the software, and there are few guarantees that the programmed functionality works. Because of the negative effects of not unit testing and the benefits of unit testing while programming in the early phases of development, many unit tests were created for the components of the server’s software.

Unit tests make it easier for the programmer to rearrange or change the components of the program while maintaining the same core functionality of the program. This type of rearrangement and modification of program components is known as *refactoring*. [37] A programmer typically refactors a program so it is more reliable and easier to maintain. If, for some reason, the program does not maintain the same core functionality after being refactored, the program should fail at least one of the unit tests. This lets the programmer know that something in the program is not functioning correctly. If the program maintained the same core functionality after being refactored, the program should pass all of the unit tests. However, the unit tests may not always catch all malfunctions in a program. Sometimes a program may pass a unit test even if the component is malfunctioning. These instances are called *false negatives*. [38] Because of false negatives, automated unit testing is not a complete replacement for manually testing a program by using it. This is why the server software was occasionally manually tested while being frequently unit tested.

5.2 User Testing

How do we know if our proposed solution fulfills the needs of sUAS operators? The only way to find out is to let sUAS operators test the system by using it and giving feedback. We should use this feedback to improve the system to better suit the needs of sUAS operators. User testing also includes usability testing, which tests how well a user can use a software system. [39]

We will not do any user testing now because the system is being actively developed. We will do user testing after the system is ready for a controlled real-world environment.

5.3 System Performance Testing

To measure the performance of the system, the system must undergo performance testing. Similar to user testing, performance testing is best done by letting sUAS operators use the system in a controlled real-world environment. However, we can simulate real world scenarios for testing the performance of the system without sUAS operators using the system. Unfortunately, testing the performance of the system in its current and incomplete state would not be beneficial, so performance testing will be delayed until more parts of the system are developed.

Chapter 6: Improved Solution

6.1 Another Look at the Required Features

In Chapter 2, there were two criteria that could not be evaluated: Maintainability and Performance.

- **Performance**: As noted in Chapter 5, we still cannot evaluate the performance of this solution because we have not tested the system for performance yet.
- **Maintainability**: We can now partially evaluate the maintainability of the system. The system is more maintainable because it utilizes the REST software architectural style. Additionally, the automated tests mentioned in Chapter 5 make it easier for the system to be maintained.

6.2 Problem with the Proposed Solution

As with any solution to the sUAS traffic management problem, a solution is useless if no one uses it. How do we get sUAS operators to use the system? We could rely on the government to create a law or regulation to force sUAS operators to use the system whenever they fly an sUAS. However, the government may not be willing to create such a law or regulation. Unfortunately, if the government is willing to create a law or regulation, it would take years for the government to write and enact a law or regulation. The only other option is to rely on sUAS operators volunteering to use the system. With this option, the system must be attractive enough for an operator to be willing to use it.

If we get sUAS operators to use our system, how do we get them to continue using our system? An sUAS operator may stop using the system after they have started using it. An sUAS operator may stop using the system because the system no longer satisfies their needs. The ability to adjust to the changing needs of sUAS operators is one of the reasons why the maintainability of the system is imperative, but it may not be enough.

As mentioned earlier in this chapter, the system is not useful if it does not attract sUAS operators to voluntarily use it. To help lessen this problem, our proposed solution will reward an sUAS operator that uses the system. The rewards create an incentive for sUAS operators to use our system to help them operate their sUAS safely and responsibly. An sUAS operator would want to use the sUTM system whenever she is operating her sUAS because she does not want miss out on the rewards or benefits of using the system.

To reward sUAS operators, the sUTM system will have a points system where an sUAS operator earns *points* when she uses the system. Eventually, the sUAS operator may be able to redeem the points she has earned for discounts or prizes. However, this feature would require participation from other businesses and organizations, so for now, the sUAS operator will receive an *achievement badge* when she earns a certain number of points. An achievement badge is a virtual item given to the user as recognition of a certain achievement.

Points and achievement badges are design elements that are often found in video games. The process of applying game design elements to a non-game context is known as *gamification*. [40] Gamification does not necessarily mean transforming something that is not a game into a game. Adding a few game elements to something that is not game does not necessarily make it a game. According to the definition proposed by Katie Salen and Eric Zimmerman in their book *Rules of Play: Game Design Fundamentals*, a game is "a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome." [41] Because there is no artificial conflict, the system in the currently proposed solution is still not a game. Users of our system are not competing against the system or each other—they are not competing against anything; they are just being rewarded for using the system.

Because many of the sUAS operations are and will be for hobby and recreational purposes, adding game elements to the sUTM system allows it to be more enjoyable to use by complementing the hobby and recreational purpose. However, commercial UAS operations do not have to use these game elements if it is inappropriate to do so. In other words, participation in the rewards system is optional.

Chapter 7: User Incentives through Gamification

To properly utilize gamification in our system, we need to design the game elements properly.

7.1 Points

An sUTM operator earns points by using the system. Table 3 shows how an sUAS operator can earn points when using the sUTM system.

Event	**Points Awarded**
Flight plan authorization was granted	5
Sign up	10
Report sUAS location	1
Randomly when using the system	0 - 10 (varies randomly)

Table 3 List of events where points are awarded to the user

Rewarding points to the user acts primarily as a tool for the positive reinforcement of certain desired behaviors. We reward the user for signing up because we want the user to feel that he has immediately benefitted from using the system, and as a result, will continue to use the system to get more benefits. We reward the user for reporting the location of his sUAS because we want him to update the location frequently, as it helps the system have the most up-to-date information about the location of the sUAS. The points given randomly provides an element of surprise that keeps the user engaged with the system. The randomly-given points also serves as a reward for just using any part of the system.

7.2 Achievement Badges

Achievement badges add a social element to the system that keeps users engaged. Achievement badges can be given by the system for reasons other than earning a certain number of points. As the system evolves, more achievement badges for different events may be added later in the sUTM system's development. Table 4 shows the current list of badges a user can earn.

Achievement badge name	**Event to earn the badge**
Aviator Level 1	Earn 100 points
Aviator Level 2	Earn 250 points
Aviator Level 3	Earn 450 points
Aviator Level 4	Earn 700 points
Aviator Level 5	Earn 1,000 points
Super Aviator Level 1	Earn 1,500 points
Super Aviator Level 2	Earn 2,100 points
Super Aviator Level 3	Earn 2,800 points
Super Aviator Level 4	Earn 3,600 points
Super Aviator Level 5	Earn 4,500 points
Flight Plan Acceptance	Get flight plan authorization granted 25 times
Flight Plan Rejection	Get flight plan authorization rejected 25 times

Table 4 List of Achievement Badges

The “Aviator” and “Super Aviator” achievement badges serve as a type of fun user ranking system that is based on the number of points a user has cumulatively earned. The system has the “Flight Plan Acceptance” and “Flight Plan Rejection” achievement badges to encourage the user to submit a flight plan even if he thinks the flight plan will be rejected.

Chapter 8: Conclusions and Future Work

Small UAS (sUAS) pose a safety hazard to both manned aircraft and people on the ground. Managing the traffic of sUAS, or UAS in general, is a challenge because of the variety and frequently-changing needs of the UAS industry. The third version of the solution to this problem proposed in this paper is a system where a server that hosts a public RESTful web service is accessible through a public application programming interface (API). Through this API, the Android mobile app we developed, along with almost any device that can access the Internet, can communicate with the server in order to utilize the server's sUAS traffic management (sUTM) functions. Because of this, software developers can integrate the server's sUTM functions, through the user of the REST API, into existing computers systems or new software. To demonstrate the capabilities of the server's REST API, we developed an Android mobile app. By separating the sUAS traffic management functionality from the visual interface(s), we increased the system's ability to adapt to unexpected changes within the UAS industry, government regulation, and sUAS operator needs. The third version of the proposed solution also includes game elements in order to make the system more attractive and engaging for sUAS operator to use.

The system in our proposed solution is currently incomplete. More development and testing are needed to create a system that can sufficiently help solve the problem of sUAS traffic management. The following is a list of the work that still needs to be done.

- **Testing**: As mentioned in Chapter 5, the system is currently too incomplete to do any useful user or performance testing. After the system is tested for performance and the fulfillment of user needs, the system will be optimized and improved based on the results of the tests.
- **Bug fixes**: Both the server software and the Android mobile app have malfunctions (or *bugs*) that should be fixed. How soon a bug will be fixed depends on how quickly the bug can be fixed and the severity of the bug.

- **Deployment**: When the system is mature enough it will be deployed for the public to use. The server will be hosted on the World Wide Web and the Android mobile phone app will be available to millions of people through the Google Play Store.

None of the versions of the solution proposed in this paper completely solve the sUAS safety problem. That is because none versions of the proposed solution were ever intended to completely solve the problem. The sUAS safety problem will continue to be an issue as long there are sUAS. The most we can do is increase sUAS safety as much as possible. Because of the variety of sUAS and sUAS usage, multiple solutions to sUAS safety problem are necessary. Although the proposed solution does not completely solve the sUAS safety problem, it can serve as a foundation for other solutions intended to increase sUAS safety.

Appendix: sUTM System REST API List

NOTE: Many of the operations belong to multiple categories. These operations are listed multiple times, once for each category they belong to.

Category: User

User management

HTTP Method and URI	Description
POST /users	Create a new user
GET /users/{username}	Get a user by username
PATCH /users/{username}	Update a user's information.
PUT /users/{username}	Update a user's information
GET /users/{username}/achievement-badges	List the achievement badges a user has earned
POST /users/auth-ticket	Get a ticket for user authentication
DELETE /users/auth-ticket	Revoke a user authentication ticket
POST /users/{username}/password	Change a user's password
GET /users/{username}/password/reset	Request to reset the password of a user
POST /users/{username}/password/reset	Confirm a request to reset the password of a user.
GET /user-id/{userId}/username	Get the username of a user

Category: UAS

Unmanned Aircraft System (UAS) management

HTTP Method and URI	Description
GET /uas	List UAS
POST /uas	Create a new UAS
GET /uas/{uasId}	Get a UAS by ID
PATCH /uas/{uasId}	Update the information of a UAS
PUT /uas/{uasId}	Update the information of a UAS
DELETE /uas/{uasId}	Delete a UAS
POST /uas/{uasId}/position	Update the position of a UAS
POST /area/uas	List UAS that are within a given area

Category: Flight Plan

Flight plan management

HTTP Method and URI	Description
GET /flight-plans	List flight plans
POST /flight-plans	Create a new flight plan
GET /flight-plans/{flightPlanId}	Get a flight plan by ID
PATCH /flight-plans/{flightPlanId}	Update the information of a flight plan
PUT /flight-plans/{flightPlanId}	Update the information of a flight plan
DELETE /flight-plans/{flightPlanId}	Delete a flight plan
GET /flight-plans/{flightPlanId}/clearance/request	Request clearance for a flight plan

Category: Flight Plan Clearance

Flight plan clearance management

Operation	Description
GET /flight-plans/{flightPlanId}/clearance/request	Request clearance for a flight plan

Category: Authentication Required

Operations that require authentication

HTTP Method and URI	Description
PATCH /users/{username}	Update a user's information.
PUT /users/{username}	Update a user's information
DELETE /users/auth-ticket	Revoke a user authentication ticket
POST /uas	Create a new UAS
PATCH /uas/{uasId}	Update the information of a UAS
PUT /uas/{uasId}	Update the information of a UAS
DELETE /uas/{uasId}	Delete a UAS
POST /uas/{uasId}/position	Update the position of a UAS
POST /flight-plans	Create a new flight plan
PATCH /flight-plans/{flightPlanId}	Update the information of a flight plan
PUT /flight-plans/{flightPlanId}	Update the information of a flight plan
DELETE /flight-plans/{flightPlanId}	Delete a flight plan
GET /flight-plans/{flightPlanId}/clearance/request	Request clearance for a flight plan
GET /test/authentication	A test GET request with required authentication
POST /test/authentication	A test POST request with required authentication
PATCH /test/authentication	A test PATCH request with required authentication
PUT /test/authentication	A test PUT request with required authentication
DELETE /test/authentication	A test DELETE request with required authentication

Category: SSL Only

The server will accept the request only if it is sent over SSL/TLS

HTTP Method and URI	Description
POST /users	Create a new user
POST /users/auth-ticket	Get a ticket for user authentication
POST /users/{username}/password	Change a user's password
POST /users/{username}/password/reset	Confirm a request to reset the password of a user.

Category: All

All API operations

HTTP Method and URI	Description
POST /users	Create a new user
GET /users/{username}	Get a user by username
PATCH /users/{username}	Update a user's information.
PUT /users/{username}	Update a user's information
GET /users/{username}/achievement-badges	List the achievement badges a user has earned
POST /users/auth-ticket	Get a ticket for user authentication
DELETE /users/auth-ticket	Revoke a user authentication ticket
POST /users/{username}/password	Change a user's password
GET /users/{username}/password/reset	Request to reset the password of a user
POST /users/{username}/password/reset	Confirm a request to reset the password of a user.
GET /user-id/{userId}/username	Get the username of a user
GET /uas	List UAS
POST /uas	Create a new UAS
GET /uas/{uasId}	Get a UAS by ID
PATCH /uas/{uasId}	Update the information of a UAS
PUT /uas/{uasId}	Update the information of a UAS
DELETE /uas/{uasId}	Delete a UAS
POST /uas/{uasId}/position	Update the position of a UAS
GET /flight-plans	List flight plans
POST /flight-plans	Create a new flight plan
GET /flight-plans/{flightPlanId}	Get a flight plan by ID
PATCH /flight-plans/{flightPlanId}	Update the information of a flight plan
PUT /flight-plans/{flightPlanId}	Update the information of a flight plan
DELETE /flight-plans/{flightPlanId}	Delete a flight plan
GET /flight-plans/{flightPlanId}/clearance/request	Request clearance for a flight plan
POST /area/uas	List UAS that are within a given area

References

[1] *Small UAS Notice of Proposed Rulemaking (NPRM)*. (2015). Retrieved December 8, 2015, from http://www.faa.gov/regulations_policies/rulemaking/recently_published/media/2120-AJ60_NPRM_2-15-2015_joint_signature.pdf

[2] Kim, Y., Jo, J., & Shaw, M. (2015). A lightweight communication architecture for small UAS Traffic Management (SUTM). *2015 Integrated Communication, Navigation and Surveillance Conference (ICNS)*, T4-6.

[3] Parrot. (n.d.). *Bebop 2*. Retrieved April 07, 2016, from https://ms03.parrot.com/789-large_default/product.jpg

[4] DJI. (n.d.). *Phantom 4*. Retrieved April 07, 2016, from http://asset1.djicdn.com/uploads/product_store_photo/image/18081/large_p2.jpg

[5] *3DR Solo*. (n.d.). Retrieved April 07, 2016, from https://s3.amazonaws.com/3dr.production/1532/large/Solo_Bundle-1.jpg

[6] Booth, D., Haas H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., & Orchard, D. (2004, February 11). Web Services Architecture. Retrieved March 31, 2015, from http://www.w3.org/TR/ws-arch/#whatis

[7] Proffitt, B. (2013, September 19). What APIs Are And Why They're Important - ReadWrite. Retrieved March 31, 2015, from http://readwrite.com/2013/09/19/api-defined

[8] REST APIs. (n.d.). Retrieved January 20, 2016, from https://dev.twitter.com/rest/public

[9] Google Maps APIs. (n.d.). Retrieved January 20, 2016, from https://developers.google.com/maps/

[10] Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* (Doctoral dissertation, University of California, Irvine).

[11] W3C. (2004, September 1). HTTP/1.1: Status Code Definitions. Retrieved April 04, 2016, from http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

[12] Hammer, E. (2012, May 11). RealtimeConf – “OAuth 2.0 - Looking Back and Moving On” by Eran Hammer. Retrieved May 22, 2015, from https://vimeo.com/52882780

[13] OAuth. (n.d.). Retrieved April 04, 2016, from http://oauth.net/

[14] RFC 6749 - The OAuth 2.0 Authorization Framework. (2012, October). Retrieved April 06, 2016, from http://tools.ietf.org/html/rfc6749

[15] Chen, E. Y., Pei, Y., Chen, S., Tian, Y., Kotcher, R., & Tague, P. (2014, November). Oauth demystified for mobile application developers. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (pp. 892-903). ACM.

[16] Hammer, E. (2010, September 29). OAuth Bearer Tokens are a Terrible Idea. Retrieved May 23, 2015, from http://hueniverse.com/2010/09/29/oauth-bearer-tokens-are-a-terrible-idea/

[17] API Glossary & Acronyms. (n.d.). Retrieved May 23, 2015, from http://apiglossary.com/

[18] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., & Stewart, L. (1999, June). RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication. Retrieved May 23, 2015, from http://tools.ietf.org/html/rfc2617

[19] Hammer, E. (2016, March 9). Hawk - HTTP Holder-Of-Key Authentication Scheme. Retrieved April 06, 2016, from https://github.com/hueniverse/hawk/blob/master/README.md

[20] Hammer, E. (2016, February 1). Oz - Web Authorization Protocol. Retrieved April 06, 2016, from https://github.com/hueniverse/oz/blob/master/README.md

[21] Mozilla Developer Network (MDN). (2015, January 19). Window.localStorage. Retrieved May 23, 2015, from https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

[22] Mozilla Developer Network (MDN). (2015, May 9). Window.sessionStorage. Retrieved May 23, 2015, from https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage

[23] Storing of credentials in browser client. (2015, February 20). Retrieved May 23, 2015, from https://github.com/hueniverse/hawk/issues/138

[24] Pub. L. No. 112-95, 126 Stat. 77 (2012)

[25] Pub. L. No. 112-95, 126 Stat. 72 (2012)

[26] Ivory, M. Y., & Hearst, M. A. (2002). Improving web site design. *Internet Computing, IEEE, 6*(2), 56-63.

[27] *Serif and sans-serif 01* [Photograph found in Wikimedia Commons]. (2007, May 4). Retrieved April 05, 2016, from https://commons.wikimedia.org/wiki/File:Serif_and_sans-serif_01.

[28] *Serif and sans-serif 02* [Photograph found in Wikimedia Commons]. (2007, May 4). Retrieved April 05, 2016, from https://commons.wikimedia.org/wiki/File:Serif_and_sans-serif_02.

[29] *Serif and sans-serif 03* [Photograph found in Wikimedia Commons]. (2007, May 4). Retrieved April 05, 2016, from https://commons.wikimedia.org/wiki/File:Serif_and_sans-serif_03.

[30] Morris, R. A., Aquilante, K., Yager, D., & Bigelow, C. (2002, May). P-13: Serifs Slow RSVP Reading at Very Small Sizes, but Don't Matter at Larger Sizes. In *SID Symposium Digest of Technical Papers* (Vol. 33, No. 1, pp. 244-247). Blackwell Publishing Ltd.

[31] Elliot, A. J. (2015). Color and psychological functioning: A review of theoretical and empirical work. *Frontiers in Psychology Front. Psychol., 6*.

[32] Preedy, V. R., & Watson, R. R. (2010). *Handbook of disease burdens and quality of life measures*. New York: Springer. doi:10.1007/978-0-387-78665-0_6467

[33] Kolenda, N. (2015, October 29). Color Psychology: The Complete Guide for Marketers. Retrieved April 05, 2016, from http://www.nickkolenda.com/color-psychology/

[34] Labrecque, L. I., & Milne, G. R. (2012). Exciting red and competent blue: the importance of color in marketing. *Journal of the Academy of Marketing Science*, *40*(5), 711-727.

[35] Chapman, A. (2014). It's okay to call them drones. *Journal of Unmanned Vehicle Systems, 02*(02), Iii-V. doi:10.1139/juvs-2014-0009

[36] Barr, E. T., Harman, M., McMinn, P., Shahbaz, M., & Yoo, S. (2015). The oracle problem in software testing: A survey. *Software Engineering, IEEE Transactions on*, *41*(5), 507-525.

[37] Griswold, W. G., & Opdyke, W. F. (2015). The Birth of Refactoring: A Retrospective on the Nature of High-Impact Software Engineering Research. *Software, IEEE*, *32*(6), 30-38.

[38] Gross, F., Fraser, G., & Zeller, A. (2012, July). Search-based system testing: high coverage, no false alarms. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis* (pp. 67-77). ACM.

[39] Dongsong, Z., & Adipat, B. (2005). Challenges, Methodologies, and Issues in the Usability Testing of Mobile Applications. *International Journal Of Human-Computer Interaction*, *18*(3), 293-308. doi:10.1207/s15327590ijhc1803_3

[40] Deterding, S., Dixon, D., Khaled, R., & Nacke, L. (2011, September). From game design elements to gamefulness: defining gamification. *In Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments* (pp. 9-15). ACM.

[41] Tekinbaş, K. S., & Zimmerman, E. (2003). *Rules of play: Game design fundamentals*.

Curriculum Vitae

The Graduate College

University of Nevada, Las Vegas

Monetta A. Shaw

Degrees:

Bachelors of Science in Computer Science, 2013

University of Nevada, Las Vegas

Awards:

2014 Nevada Space Grant Scholarship

Thesis Title: A Web-Based Solution for Small Unmanned Aircraft System (sUAS) Traffic Management