University of Victoria

Computer Science Co-op

Work Term Report

Summer 2019

Comparison of Authentication and Authorization Methods Pertaining the XXXXXXX RESTful API

XXXXXXX

Web Development

Victoria, BC, Canada

Nathan Jenkins-Boale

V00851969

2

Computer Science

njboale@uvic.ca

August 30th 2019

Supervisor's Approval: To be completed by Co-op Employer

This report will be handled by UVic Co-op staff and will be read by one assigned report marker who may be a co-op staff member within the Engineering and Computer Science/Math Co-operative Education Program, or a UVic faculty member or teaching assistant. The report will be either returned to the student or, subject to the student's right to appeal a grade, held for one year after which it will be destroyed.

I approve the release of this report to the University of Victoria for evaluation purposes only.

Signature: _____ Position: _____ Date: _____

Name (print): _____ E-Mail: _____

For (Company Name)_____

Abstract

The following report outlines methods of user authorization and authentication for accessing a RESTful API. The methods to be discussed are: API keys, basic auth, and OAuth.

Basic auth and key signing methods face significant security concerns since both these methods require that sensitive information be sent on each request. This increases the probability of data interception and decryption. If this information is obtained, it may be used to perform malicious requests to alter data or collect user sensitive information.

OAuth was chosen as the authorization / authentication flow as it provides the best functionality for the XXXXXXX needs. It allows users to be uniquely identified with a token that may not be altered, insuring that the user performing the request is who they say they are. In addition, since users can be uniquely identified it can distinguish between different types of user accounts and evaluate if they have access rights to the requested resource. Finally, the token may contain an expiry data, meaning that if the token is stolen by a third party it will only be valid for a limited amount of time.

Report Specifications:

Audience

This report is intended for individuals who want an overview of three well tested, and frequently used authentication and authorization methods for accessing an API. The methods to be discussed are : API keys, basic auth, OAuth, and the specific XXXXXXX authentication strategy following the OAuth strategy with JSON web tokens. The report is designed for someone who wants to broaden their knowledge of these different authentication and authorization strategies, their potential security risks and ways to mitigate these issues.

Prerequisites

The reader should have a working knowledge of web services and a simplistic understanding for the client - server model for handling requests. They should also have knowledge of the different types of requests and an overview of how these requests are handled by a server. This report also assumes that the reader has a general overview of web system architecture and are able to follow the flow of data through a system.

Purpose

The purpose of this report is to give an overview of different production ready techniques for user authentication and authorization while accessing a RESTful API. This report will identify issues with each of the methods and make an informed decision for which method to implement for XXXXXXX services. It will provide the reader with a broad understanding of what context each of these strategies should be used in and allow them to make their own decision for implementing new software.

Table of Contents

List of Figures

Introduction:

The following report will outline a number of strategies used to authenticate and authorize users accessing a RESTful API. It will identify the strengths and weaknesses of each strategy and outline a solution following the OAuth Strategy.

XXXXXXX is a young business located in Victoria. The company creates workplace fitness products that aim to promote a healthier office lifestyle. XXXXXXX is developing two hardware products: a standing desk and a bike that sits under the desk. These products connect directly to the users' phone or computer and allow them to make changes and track progress. The company promotes a lively and resourceful environment where members work towards solutions in a collaborative environment.

RESTful APIs have gained popularity over the years because of their lightweight footprint and ability to interact with many different services identically [1]. In this way a mobile app, website and any number of other components may obtain identical data based on the same requests. This architecture decreases code repetition and insures that all requests for data pass through the same portal.

> *A RESTful API is an application programming interface that used different HTTP requests to manage data.It uses GET, PUT, POST and DELETE requests to interact with the data [6].*

All XXXXXXX services connect to a single database through a shared application programming interface. When a user creates a request, the API determines the requests validity. This step is known as authentication and authorization.

Authentication: verifying that the given user that is performing the request is actually this user.

Authorization: verifying that requested action is allowed for this given user.

This report will analyze different methods to provide security and different levels of access for different types of users.

The reader should consider the following report limitations. The report does not describe the inner workings of the system and how these components work on a low level. Only the flow and structure of the authentication strategies at high level are discussed. For example, the report does not discuss encryption and decryption techniques, nor does it address how JSON web tokens are mathematically implemented. It treats many aspects of the process as black box and expects that they will work as intended without failure. It also does not discuss in full all potential exploits of the system. It expects that the individual features implemented including HTTPS, or encryption to work as expected, without exploitation when in reality they may not be fully secure. This If the reader wishes to understand in more depth the inner mathematical workings or additional exploitation methods they must consult an additional resource.

Main Body

The intended structure at XXXXXXX is as follows.

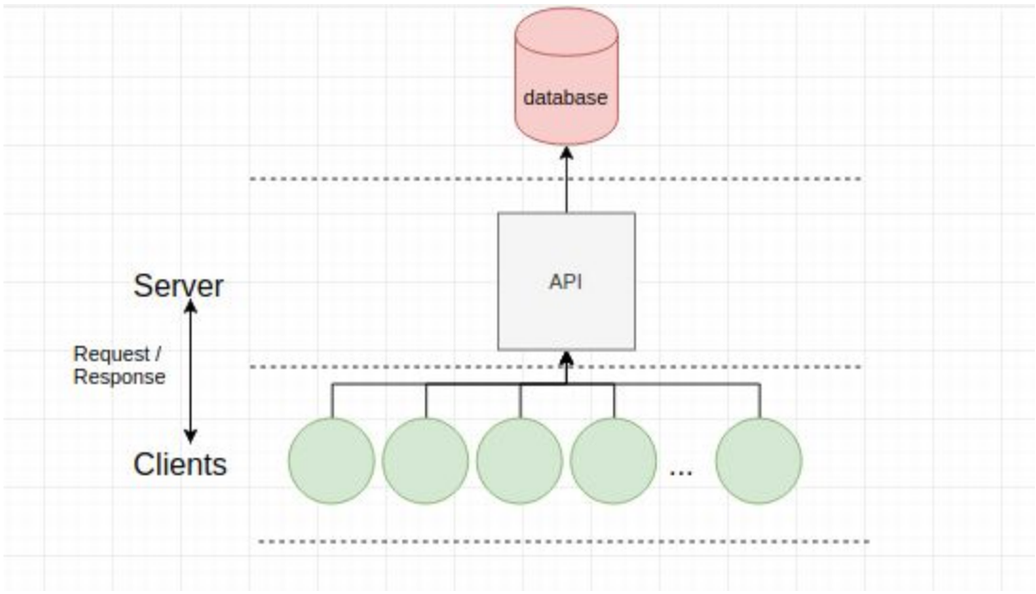

FIGURE 1 : XXXXXXX STRUCTURE

There is a RESTful API that exists between the XXXXXXX clients and its database. This application programming interface takes in requests from the clients (other XXXXXXX services) authenticates and authorizes them. If these requests are valid then the intended action is performed. This “action” is usually manipulating some kind of data within the database. This could include performing any of the four types of requests : get, put, post or delete.
This is the indent and the design of the current architecture, however, there are several implementation issues that prevent this functionality.

This report will focus on solutions for authenticating and authorizing unique user requests. In the context of the report authenticating refers to identifying that the user performing the request is the actual user, and authorizing refers to identifying if the given user has permission to perform this action.

The XXXXXXX system provides some form of authentication and authorization requests by API key signing. This concept works in the following way. The API securely stores a list of keys for

valid XXXXXXX services. When a client creates a request the key is passed along with the request. When the API receives the request it validates if the key is apart of its list and grants access to its resources [2].

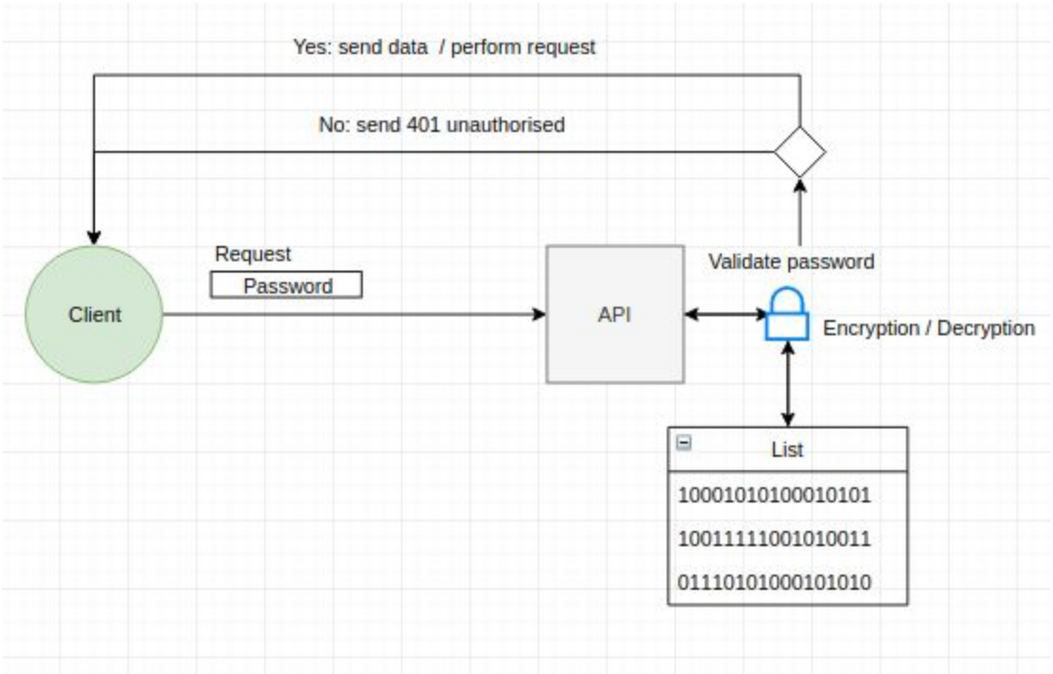

FIGURE 2 : API KEY SIGNING

This system provides authentication that the request came from a valid XXXXXXX service. However, there are some significant problems with the current implementation. For starters, it is obvious that this method is susceptible to man in the middle attacks. That is, the key sent by the client in a valid request may be intersected and saved before it reaches the API. This key may then be used on subsequent requests to perform malicious actions like retrieval of user information or deleting data [2].

In addition, where the system does provide authorization and authentication for an entire XXXXXXX clients, it does not provide this for unique users of the client. That is, a request may come from an XXXXXXX client, however, there is no authentication process to verify that the information requested belongs to that user. For example, when a user wants to access information pertaining to themselves, they send the key validating the request and a unique identifier for the user along with whatever other information that is necessary. If the user changes

the unique identifier the system will still authenticate the request, since it is coming from a valid XXXXXXX service. In this way a user could obtain information that does not belong to them.

To fix these problems, we need a way to insure that all request traffic is coming from a valid source, authenticate that the given user is providing the correct identity, and guarantee the user is authorized to perform the requested action. That is, different types of users may perform a subset of all requests. An admin for example may be allowed to manipulate certain data that a user may not.

The following report will outline different methods to achieve the API keys, Basic Auth, OAuth

Basic Auth:

In this architecture the user supplies their unique identification and password. These two parameters are concatenated with a colon separating and encoded using base64 (to insure the request url is safe) then passed in the request header to the API. The API should then verify if the user exists in the system with the given password they provided. After which we can check the role of the unique user and perform the request if everything is successful [3].

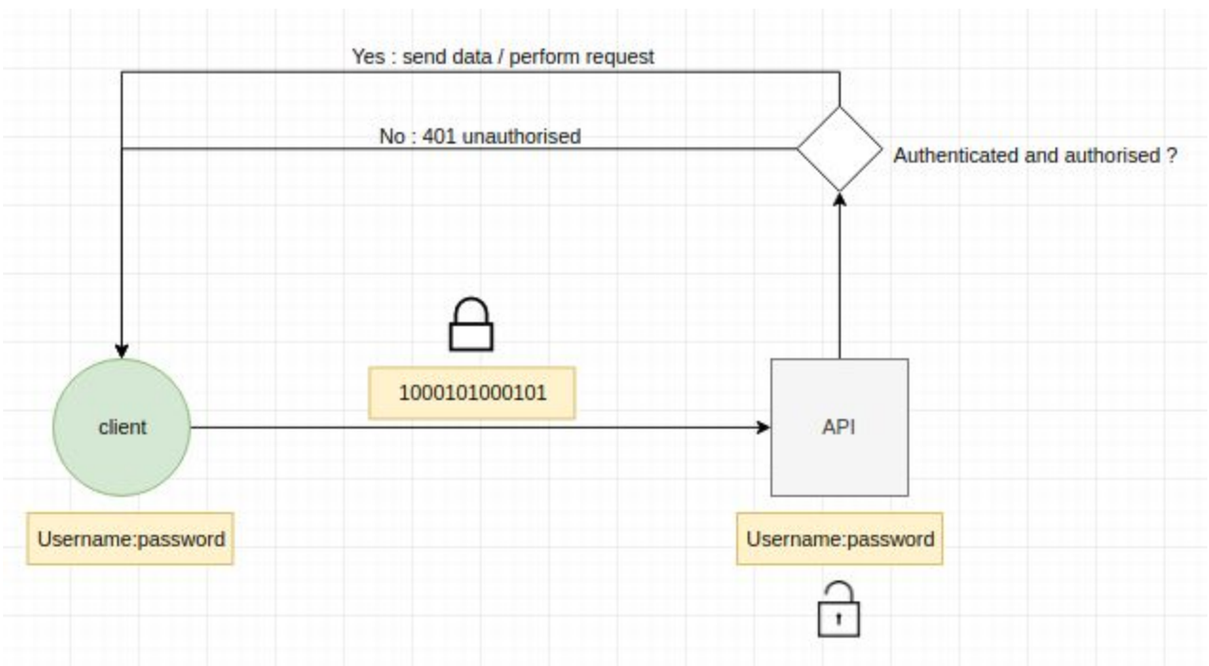

FIGURE 3 : BASIC AUTH

This system fixes the issue of user authentication and user authorization. The given user is authenticated by the unique identifier and password in the request, and authorized by checking the users' role and verifying that the information they request allowed for the role.

It should be noted that the system is vulnerable by man in the middle attacks. Someone may intercept a request and decode the unique identifier and password to be used on subsequent requests. This is a huge security consideration, for this reason it is required that we do all requests involving username and password with HTTPS.

> *What is HTTPS: At a high level, https allows for encrypted information to be sent from the client to the server. This works in the following steps: the client and server agree to an encryption method, the client verifies the server's identity, client encrypts its information using the public key provided by the server, and the server decrypts the information using its private key. The information passed over the connection is now encrypted and red uces the risk of man in the middle attacks [5].*

The number one drawback to this method is that the password and username of the user must be supplied with every request. Even if HTTPS is used with the password and username encrypted, this sensitive information must still be sent on each request, and may be susceptible for credential theft by the following ways.

1. Brute force private key / lost private key: If the private key is obtained, it can be used to decrypt the information

2. Password / username browser caching: which can be used with a cross site request forgery, where the user performs unintended actions by hidden requests on the page or through malicious links they click.

3. Rely on HTTPS to be secure and for all requests to HTTPS: if not properly implemented a request may be hijacked and SSLstripped to convert it into an HTTP request, at this point a man in the middle attack can be performed to obtain the credentials

Basic auth with HTTPS is mostly secure with some small considerations mentioned above. The most important of which is continuously sending credential information with each request. A small oversight in any route could lead to a serious data leak, of users passwords and usernames.

OAuth:

The OAuth procedure allows only a single request containing username and password to be performed. All subsequent requests are granted using the token.

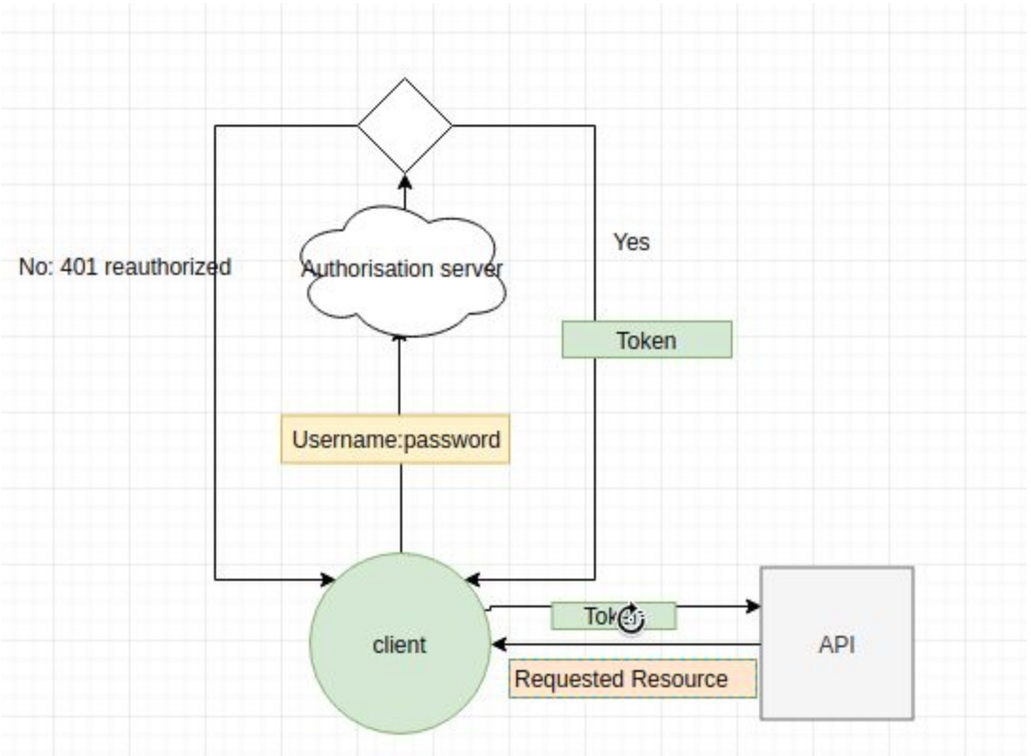

FIGURE 4 : OAUTH

OAuth works in the following way: the client (the web browser) asks for a protected resource from the API. The API will deny this request prompting the user to enter their login details. The request, along with the login details is sent as a grant to the authorization server. The authorization server

will determine if the user request is valid. That is, is the user request within the scope of what they are allowed to access (authorized) and are their login credentials valid (authenticated)? If either of these are not true it will respond with a 401 unauthorised. Otherwise the server will respond with an access token. This access token is stored by the client and sent with every subsequent request for their specific details [4].

In this way the user only provides their login credentials once, a significant improvement to both API keys and Basic auth in which sensitive system or user information is sent on each request.

This method of authentication and authorization provides several useful features. For one, the token provided by the authorization server provides a token that is unique to the user and to the request. So in this way the API can uniquely distinguish which user is performing the request and exactly what they are asking the API to do. The API can make an informed decision based on the users role (user or administrator) whether they are allowed to perform this request and if they are, execute it for this specific user. The users' details would no longer be required in the request, they would be provided within the token itself.

This solution reduces the risk of a man in the middle attack since login credentials are only sent once to a secure server.

As an added bonus, many large companies including facebook, google, twitter and so on that users may already have accounts with can also be used as a login data. Their secure authorization servers may be used to gain an access token and exchange this token with XXXXXXX api for an XXXXXXX token. This procedure is outlined below.

XXXXXXX solution:

This report proposes that XXXXXXX incorporates the OAuth flow into their API services. The following system outlines how OAuth flow was incorporated into the XXXXXXX API as a method to secure data, and insure that each given user may only access their respective data when necessary.

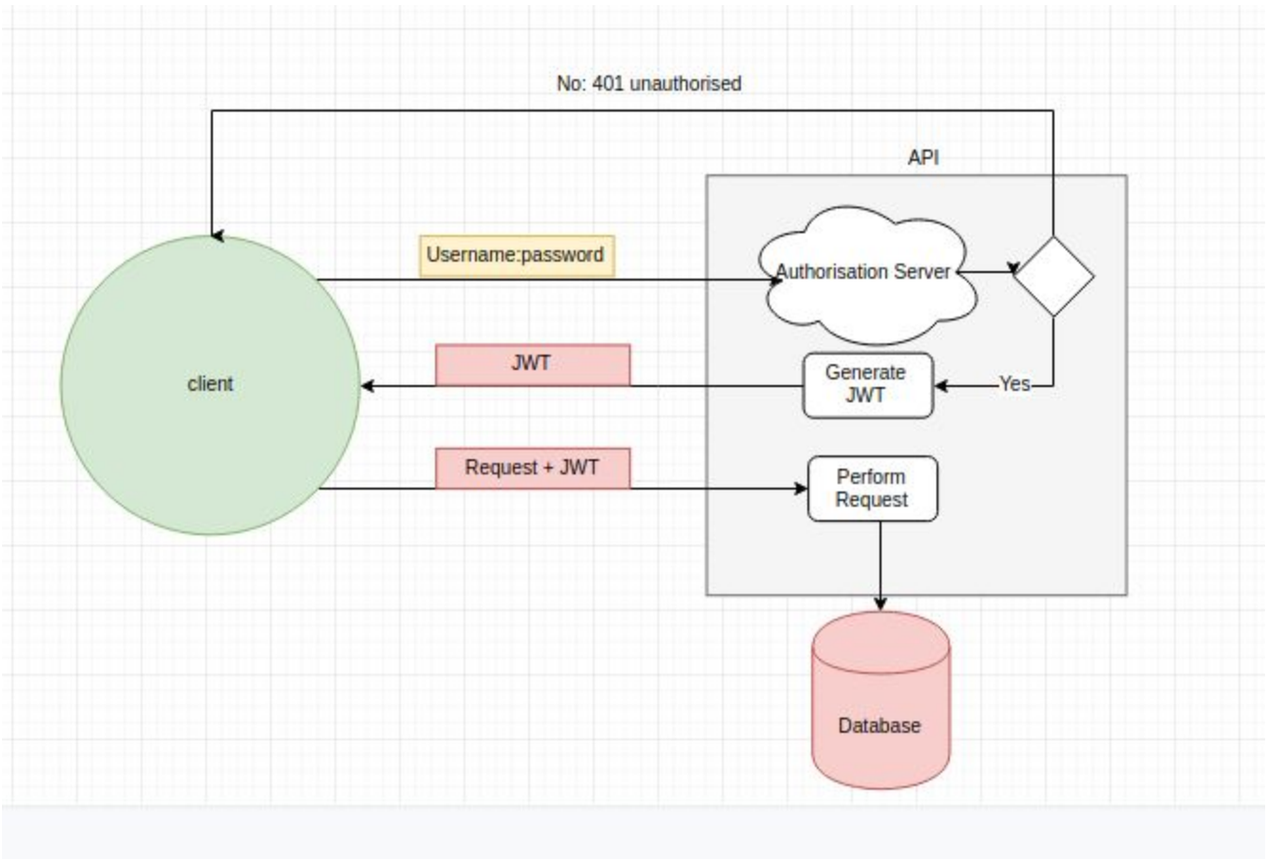

FIGURE 5 : XXXXXXX OAUTH FLOW

The client will send a request for resources to the API, if they do not provide a JSON web token or the token is not valid the API will return a 401 unauthorized. To generate a valid JSON token the user will supply a unique identifier and a password to the API, the API will validate the user. It will return a unique token for the user, the token will be provided with every request to retrieve valid resources.

JSON web tokens

The JSON web tokens are extremely powerful. It is made up of the following things, header, payload and a signature. The header specifies the encryption algorithm and type of token, the payload contains the information that will be interpreted by the API. It contains the sub, the unique identifier for the user in the database, and finally it contains the signature. The signature ensures that this token has been issued by the API and that it has not been altered along the way. This insures protection from man in the middle attacks. If someone alters the payload, the token will no longer be accepted by the API and return a 401 unauthorised.

The payload may also include a time issued and expired to insure that if a JSON web token get compromised, that is it gets stolen from a third party, this party may only use the token until it expires. In this way tokens are made more secure if they are compromised.

Example for user 1:

1. User 1 provides there login details
2. API authenticates the details
3. API creates a JWT for the user with a time created and a time that it will expire.
4. The API returns the JWT to the user
5. The user sends their a request with the JWT to access resources
6. The API decodes the JWT, authorizes and returns the information for the user identifier in the JWT payload

Since we supply the user identification in the payload, and we know the payload has not been altered, Then we will return information for the user in the payload.

Conclusions:

OAuth flow with JWT token was chosen as the best alternative to authenticate and authorize user requests in the API. This method was chosen because it provides a higher level of security and precision than key signing or basic authentication. OAuth with JWT supports these benefits by relying on the token as the means of user authentication and authorization. The token does not contain any sensitive information like basic authentication method or key signing requires. For key signing, if the key is stolen, this person may perform any action as any user within the single XXXXXXX resource they obtained the key from. Alternatively, if a basic auth request is intercepted and decrypted this person will be able to obtain the login credentials for that specific user. Since many users have identical passwords over many sites this is a huge security concern.

For these reasons OAuth with JWT was chosen to authenticate and authorize user requests. On each request a JWT token is sent to the API. This token does not contain any sensitive information, only a unique identifier for the user. In addition, each token has an expiry date, so if the token is intercepted or lost this token will only be useful for a limited amount of time before it is considered invalid.

In addition, OAuth flow is already used as an authentication method for google, facebook and others. So it is easier to implement login and authenticate with a trusted third party. Preventing the user from entering an email or password into the system. For all these reasons, OAuth provides the best solution to providing security and accuracy by authenticating and authorizing unique and dependent user requests.

Acknowledgements

Supervisor: Sergio Perez

Web developer partner: Nolan Kurylo (co-op student)

References

G. Levin, “The Rise of REST API,” REST API and Beyond, 24-Feb-2018. [Online]. Available: https://blog.restcase.com/the-rise-of-rest-api/.

“Generic Security Service Application Program Interface (GSS-API) Authentication and Key Exchange for the Secure Shell (SSH) Protocol,” IETF Tools. [Online]. Available: https://tools.ietf.org/html/rfc4462.

“The 'Basic' HTTP Authentication Scheme,” IETF Tools. [Online]. Available: https://tools.ietf.org/html/rfc7617.

“The OAuth 2.0 Authorization Framework,” IETF Tools. [Online]. Available: https://tools.ietf.org/html/rfc6749.

“What is HTTPS? - Definition from Techopedia,” Techopedia.com. [Online]. Available: https://www.techopedia.com/definition/5361/hypertext-transport-protocol-secure-https.

“What is RESTful API? - Definition from WhatIs.com,” SearchAppArchitecture. [Online]. Available: https://searchmicroservices.techtarget.com/definition/RESTful-API.