**RealTime STEAM Documentation**

**Design Documentation**

**Install Instructions:**

To deploy this application to Heroku the user needs to create an account. Once the account is created the user can go to <https://github.com/2019-Arizona-Opportunity-Hack/Team-6> and click the “Deploy to Heroku” button that is located in the README.md portion at the bottom of the page. Once the link loads, give the application a name and press the “Deploy app” button at the bottom of the page.

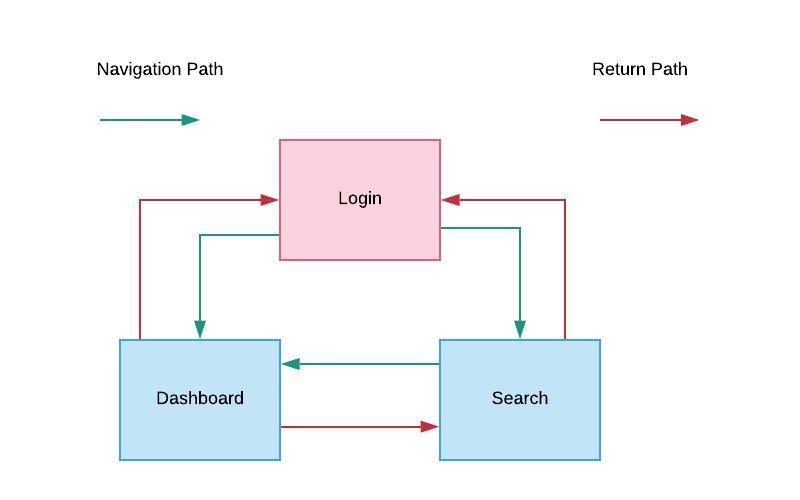
**General Technical Approach:**

This web application will utilize APIs (refer to documentation) to search the internet for trends between pop culture and STEM/STEAM themes. The user will input a search term, for example movies, and see the latest trends for movies. The user will then be able to click on an item in the returned table to see detailed data about that item. The data will be inputted onto a screen in multiple graphs, charts, and tables to accurately showcase the popularity of the searched item, as well as the most popular STEM/STEAM themes that are talked about within that topic. There will also be a generate report button that allows the user to have all the data presented on the page to be downloaded into a .csv file on their harddrive. The application also features a login/logout system that will use the proposed authentication service FusionAuth (refer to proposal information below) to protect the data from unwanted sources.

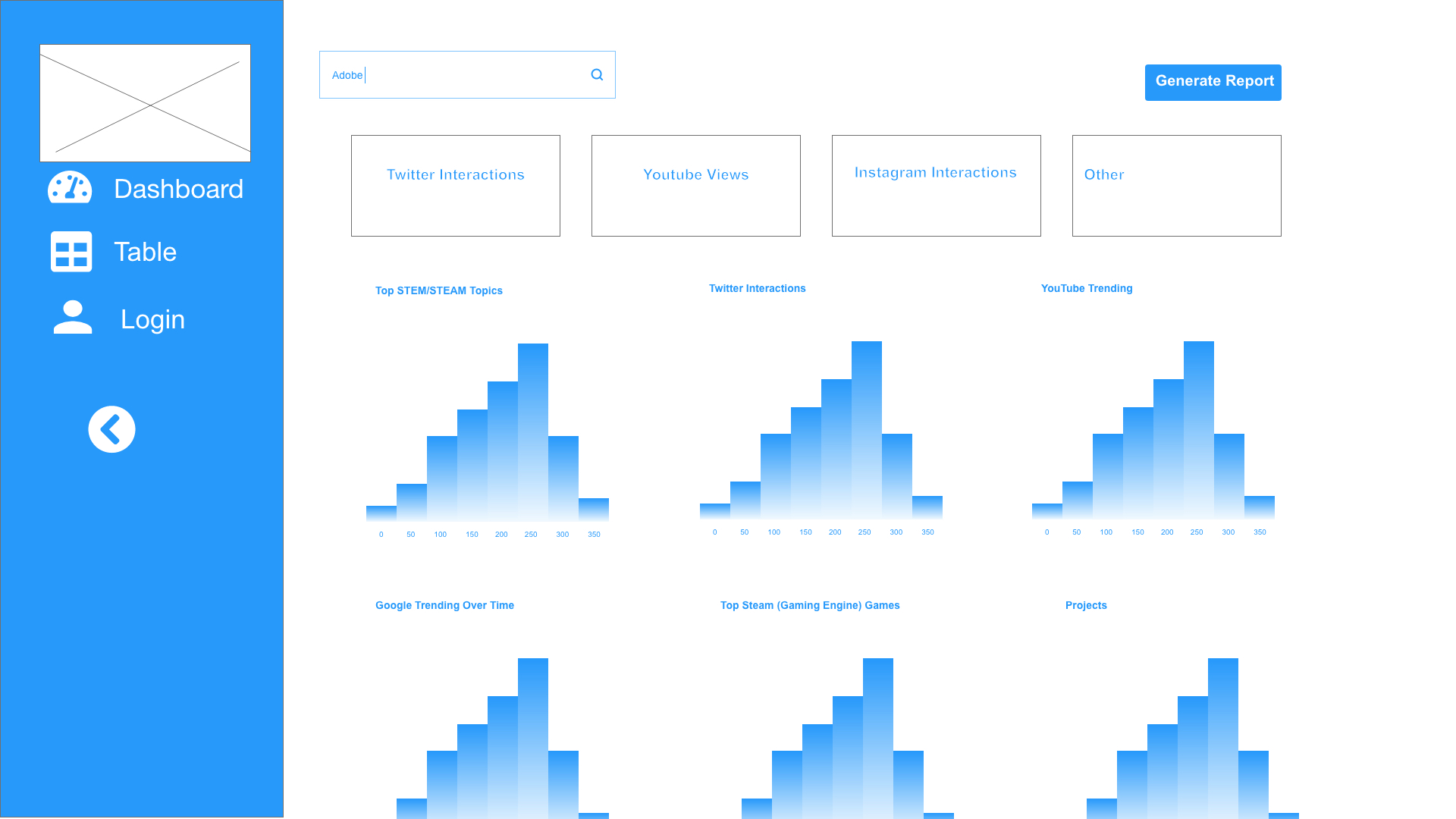
**Key Technical Design Decisions:**

The web framework Flask was chosen so that we could choose which components we wanted to use as well as being a lighter framework.

**Sitemap Diagram:**

****

**User Interface Diagrams:**

****

index.html - page with all of the detailed graph information on the search from the table.html page

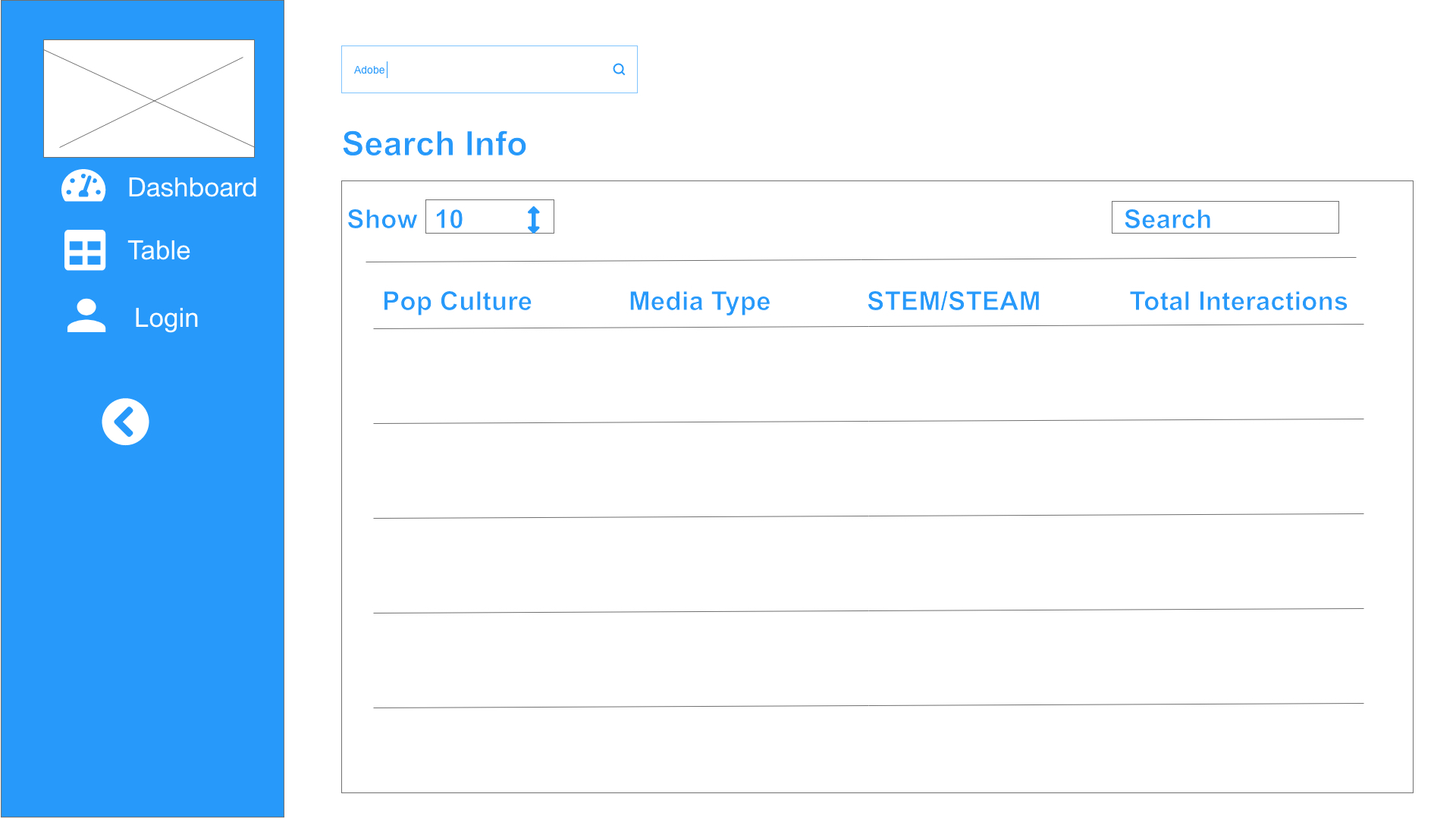
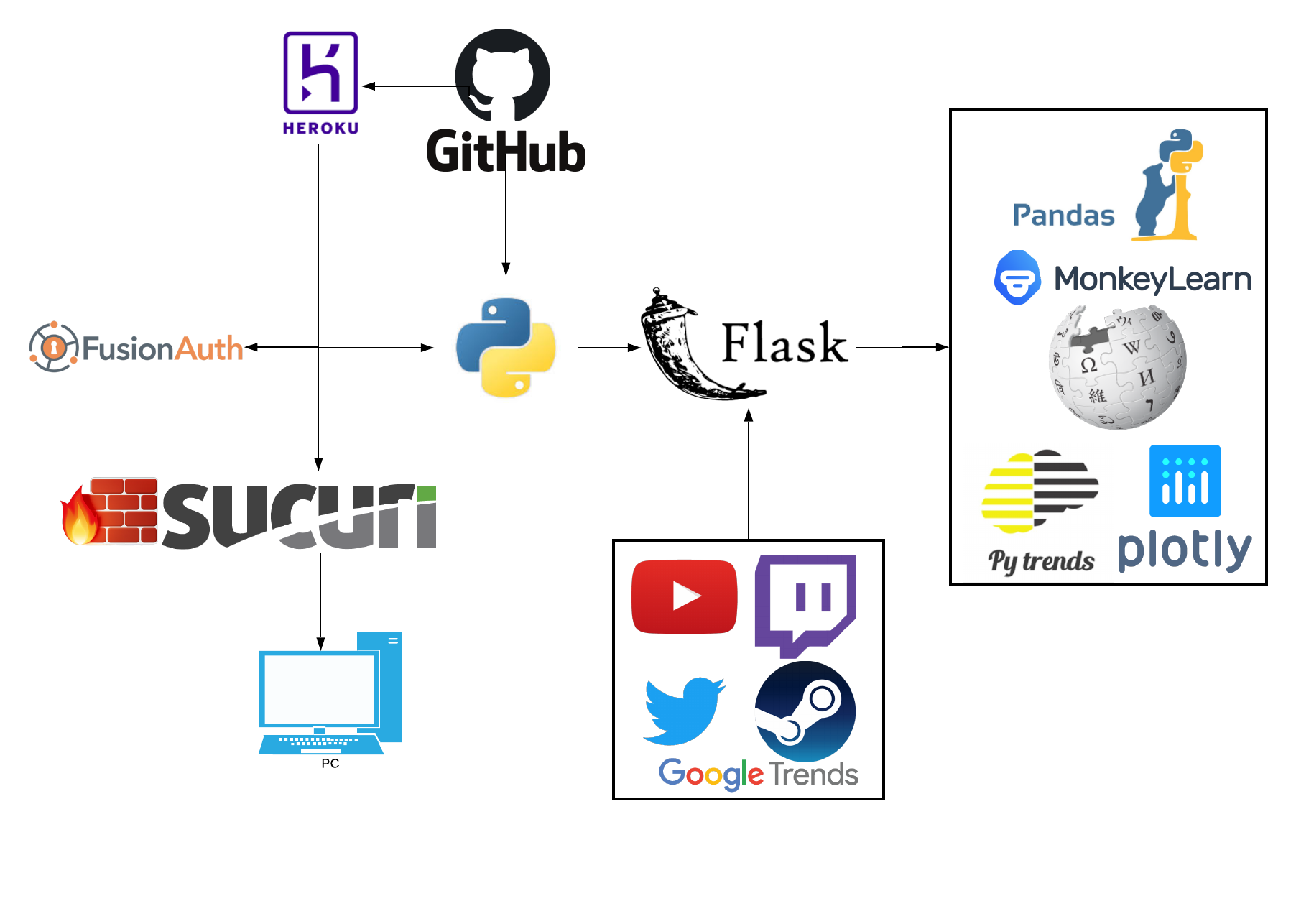
****

table.html - page where the users inputs a search (movies) and receives the top results from that search

****

login.html - where the user will log in to the system

**Class Diagrams:**



**Service API Design:**

Here are the Third Party Service Interface API’s being consumed with this product

PyTrends

Allows simple interface for automating downloads of csv reports from Google Trends. \* Main feature is to help trick google into thinking the script is actually a browser.

pip install pytrends

## Requirements

* Written for both Python 2.7+ and Python 3.3+
* Requires Requests, lxml, Pandas

## API

### Connect to Google

from pytrends.request import TrendReq

pytrends = TrendReq(hl='en-US', tz=360)

## API Methods

The following API methods are available:

* [Interest Over Time](https://pypi.org/project/pytrends/#interest-over-time): returns historical, indexed data for when the keyword was searched most as shown on Google Trends' Interest Over Time section.
* [Historical Hourly Interest](https://pypi.org/project/pytrends/#historical-hourly-interest): returns historical, indexed, hourly data for when the keyword was searched most as shown on Google Trends' Interest Over Time section. It sends multiple requests to Google, each retrieving one week of hourly data. It seems like this would be the only way to get historical, hourly data.
* [Interest by Region](https://pypi.org/project/pytrends/#interest-by-region): returns data for where the keyword is most searched as shown on Google Trends' Interest by Region section.
* [Related Topics](https://pypi.org/project/pytrends/#related-topics): returns data for the related keywords to a provided keyword shown on Google Trends' Related Topics section.
* [Related Queries](https://pypi.org/project/pytrends/#related-queries): returns data for the related keywords to a provided keyword shown on Google Trends' Related Queries section.
* [Trending Searches](https://pypi.org/project/pytrends/#trending-searches): returns data for latest trending searches shown on Google Trends' Trending Searches section.
* [Top Charts](https://pypi.org/project/pytrends/#top-charts): returns the data for a given topic shown in Google Trends' Top Charts section.
* [Suggestions](https://pypi.org/project/pytrends/#suggestions): returns a list of additional suggested keywords that can be used to refine a trend search.

Wikipedia

The Wikipedia API is a [web service](https://en.wikipedia.org/wiki/Web_service) that allows access to some wiki-features like authentication, page operations, and search. It can provide [meta information](https://www.mediawiki.org/wiki/Special:MyLanguage/API:Meta) about the wiki and the logged-in user. More information: <https://en.wikipedia.org/w/api.php>. Highlight information listed below:

**format**

The format of the output.

[**json**](https://en.wikipedia.org/w/api.php?action=help&modules=json) Output data in JSON format.

[**jsonfm**](https://en.wikipedia.org/w/api.php?action=help&modules=jsonfm) Output data in JSON format (pretty-print in HTML).

[**none**](https://en.wikipedia.org/w/api.php?action=help&modules=none) Output nothing.

[**php**](https://en.wikipedia.org/w/api.php?action=help&modules=php) Output data in serialized PHP format.

[**phpfm**](https://en.wikipedia.org/w/api.php?action=help&modules=phpfm) Output data in serialized PHP format (pretty-print in HTML).

[**rawfm**](https://en.wikipedia.org/w/api.php?action=help&modules=rawfm) Output data, including debugging elements, in JSON format (pretty-print in HTML).

[**xml**](https://en.wikipedia.org/w/api.php?action=help&modules=xml) Output data in XML format.

[**xmlfm**](https://en.wikipedia.org/w/api.php?action=help&modules=xmlfm) Output data in XML format (pretty-print in HTML).

MonkeyLearn

Welcome to the MonkeyLearn API v3 documentation. Find more information here: <https://monkeylearn.com/api/v3/#introduction>

Here you’ll find everything you need to access the MonkeyLearn API. You can use it to classify, extract, or manage your custom models programmatically.

We provide a [REST](http://en.wikipedia.org/wiki/Representational_state_transfer) API that you can access directly or through a client library.

Currently, we officially support client libraries for the following programming languages:

* [Python](https://monkeylearn.com/api/v3/#python)
* [Ruby](https://monkeylearn.com/api/v3/#ruby)
* [PHP](https://monkeylearn.com/api/v3/#php)
* [Javascript](https://monkeylearn.com/api/v3/#javascript)
* [Java](https://monkeylearn.com/api/v3/#java)

**Security Proposal:**

**OWASP Top 10**

As far as security goes for this web-based application, it should at least consider and try to satisfy OWASP’s Top 10 security risks and mitigation strategies.

1. Injection

*Occurs when an attacker sends untrusted data to an interpreter that is executed without authorization as a command.*

*To easily detect injection flaws, application security testing should be done within the scope of parameterized queries.*

2. Broken Authentication and Session Management

*Because of incorrectly configured user and session authentication, this leaves a vulnerability open for hackers to track passwords, keys, session tokens, or even take control of users’ accounts to assume their identities.*

*To reduce the risk of compromised accounts, multi-factor authentication, like FIDO, should be used.*

3. Sensitive Data Exposure

*Applications and APIs that contain sensitive data that are not securely protected are left open to hackers to access such information and use it to commit fraud, steal identities, etc.*

*To comply with encryption data protection regulations, data at rest and in transit should be encrypted.*

4. XML External Security

T*his refers to poorly configured XML processors that evaluate external references within XML documents. Attackers can use these external entities to their advantage and conduct attacks including remote code execution, disclosing internal files, SMB file shares, etc.*

*To discover this issue and further mitigate, static application security testing can be done through inspecting dependencies and configuration.*

5. Broken Access Control

*Because of improperly configured and missing restrictions on authenticated users, hackers are able to access unauthorized functionality and data.*

*Unlike other testing methods where they can only detect where access controls are missing, penetration testing is essential in detecting for non-functional access controls.*

6. Security Misconfiguration

*Refers to the improper implementation of controls that are intended to keep application data safe.*

*Dynamic application security testing (DAST) can help detect these misconfigurations, such as leaky APIs.*

7. Cross-Site Scripting

*This allows for hackers to inject client-side scripts into the application, such as redirecting users to malicious websites instead.*

*To help programmers prevent cross-site scripting, developer training complement security testing along with best coding practices should be implemented.*

8. Insecure Deserialization

*This enables attackers to execute code in the application remotely, tamper or delete serialized, or written-to-disk, objects, conduct injection attacks, and elevate privileges.*

*While application security tools can function the same way to detect these vulnerabilities, penetration testing is more frequently used.*

9. Using Components with Known Vulnerabilities

*Developers are frequently in the dark about which open source and third-party content are in their applications, making it difficult to keep these components up to date when new vulnerabilities are discovered. Attackers can then exploit these vulnerable components to take over the server and steal data.*

*A software composition analysis should be done at the same time as a static analysis to identify insecure versions of components.*

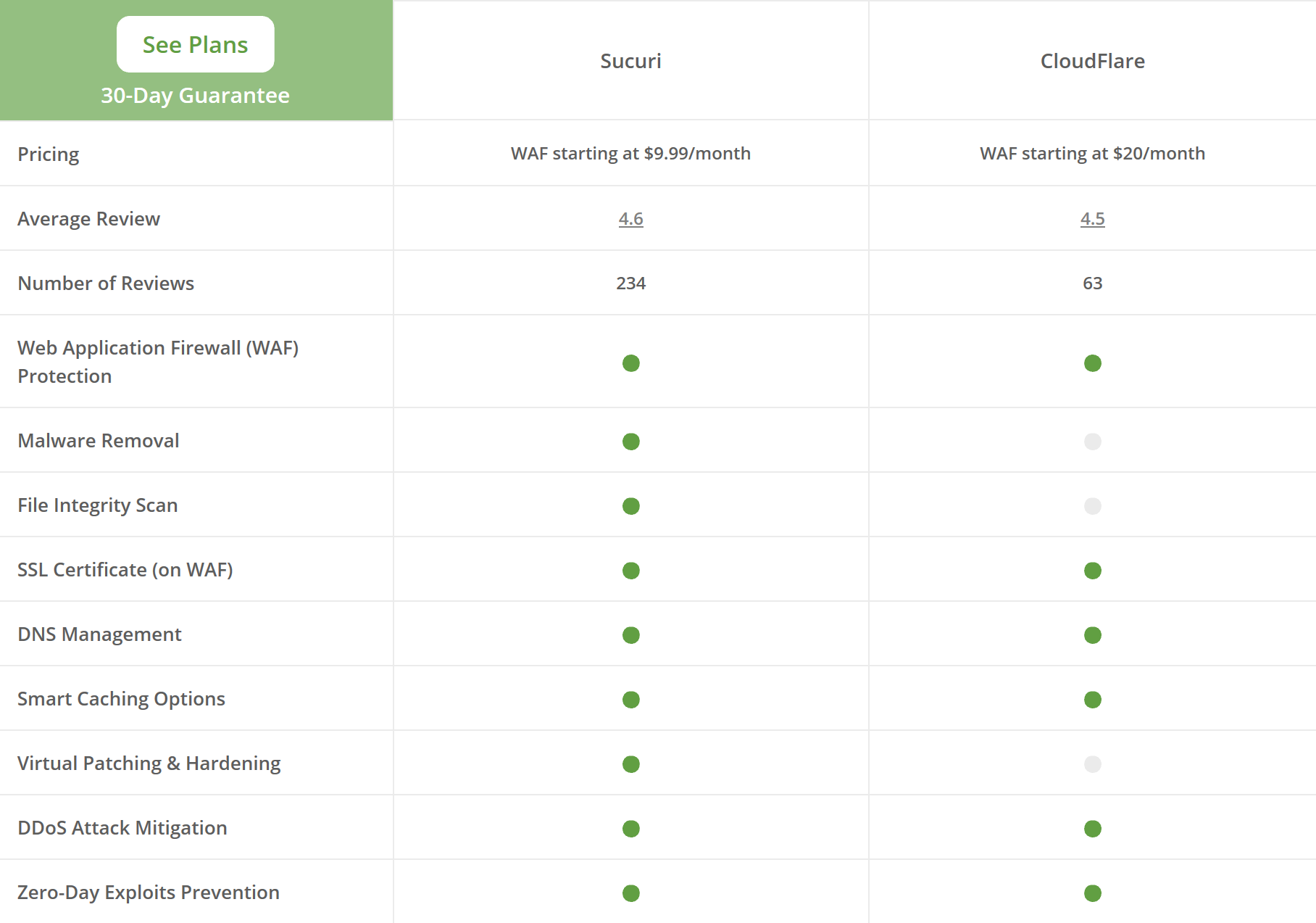
10. Insufficient Logging and Monitoring

*The time to detect a breach is frequently measured in weeks or months. Attackers are then able pivot to other systems and maintain persistent threats because of insufficient logging and ineffective integration with security incident response systems.*

*Penetration testing can help detect if there is any insufficient monitoring within the network.*

**Web Application Firewall (WAF)**

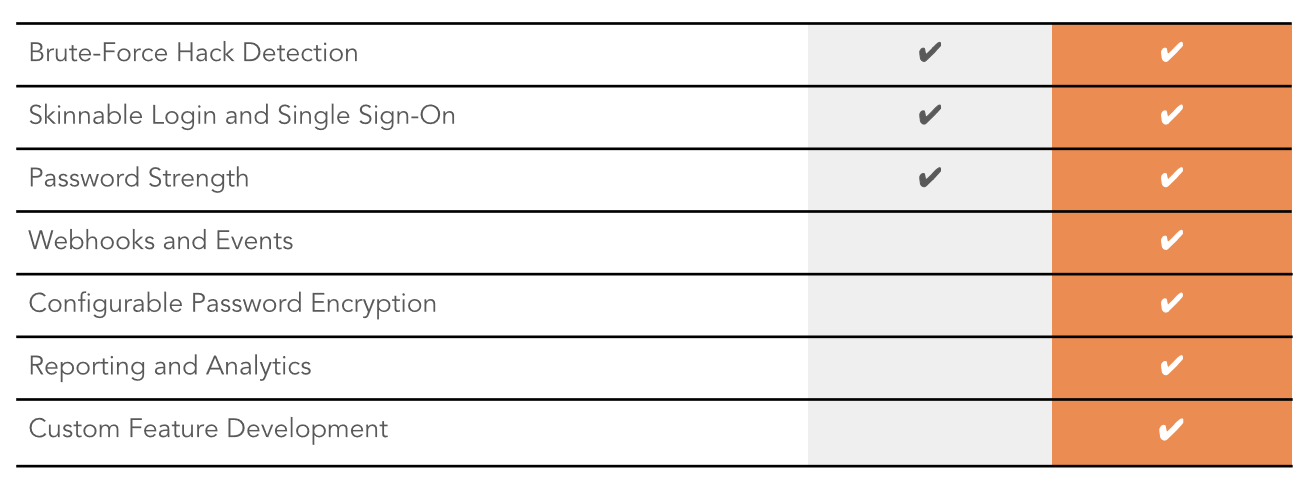
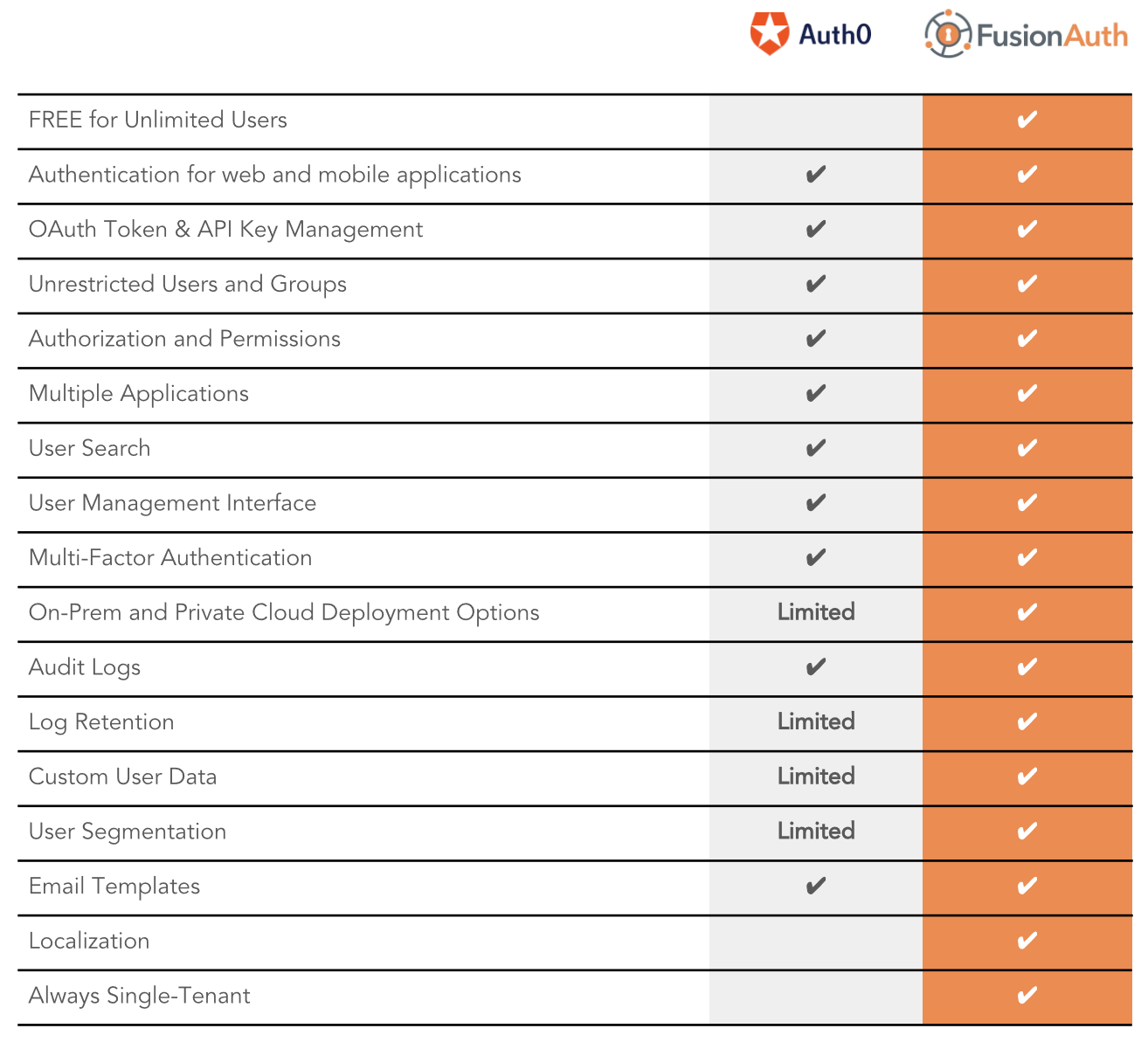
A Web Application Firewall (WAF) helps protect against application layer attacks (layer 7 of the OSI model) such as malicious HTTP traffic. A WAF is placed as a filtration barrier between the targeted server and users and acts against attacks like cross site forgery, cross site scripting, and SQL injections. The WAF should implement a combination of whitelisting and blacklisting to provide the most security to the application. Whitelist WAFs will only admit traffic that has been pre-approved whereas blacklist WAFs will use pre-set signatures to block web traffic that is clearly malicious. Below is a comparison of two of the top cloud-based WAFs with recommendations.



Sucuri and CloudFlare are both top providers in WAFs and while both are highly rated, Sucuri offers more capabilities such as malware removed and virtual patching and hardening at a lower rate than CloudFlare.

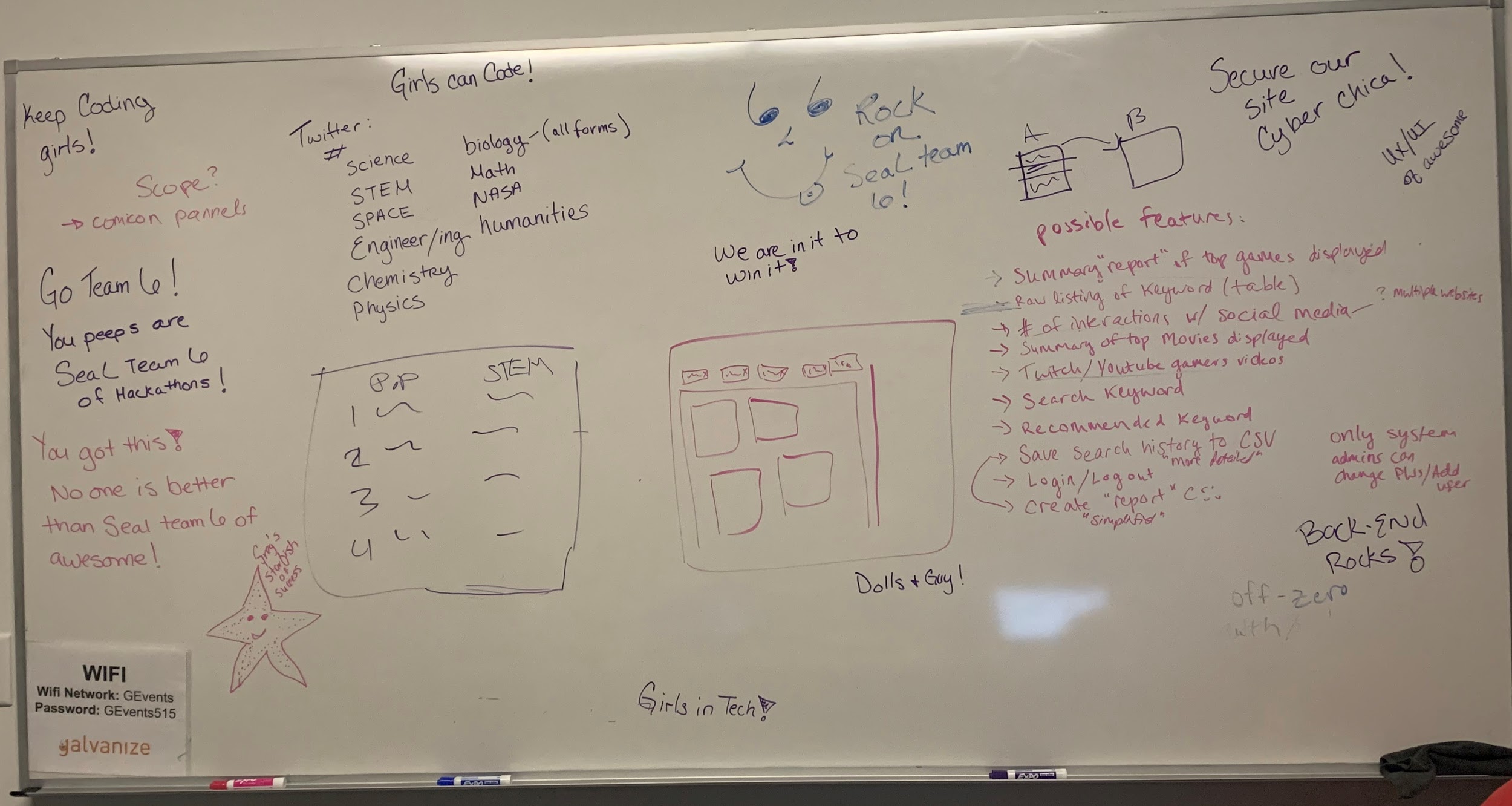
**User Authentication**

User authentication covers all the human-to-computer interactions that require the user to login to access certain information. There are currently many ways to authenticate a user, and with the help of a web-scale identity (CIAM) platform, user authentication security can be more enforced. Two CIAM platforms that’ll be looked at are FusionAuth and Auth0. Both provide user authentication services on top of web applications.



FusionAuth provides the core functions a CIAM should provide such as registration, login, single sign-on (SSO), multi-factor authentication (MFA), password hacking, password constraints, password reset, and more. It also provides localization features, reporting, analytics, user segmentation, user search, and a user management UI. On top of all that, it is free for unlimited users and is usable for authentication on web and mobile applications.

**Other Documentation:**



Initial Planning