DATAVA

# Hubspot Integration

Maksym Bondar, Dustin Choat, David Esposito, Jordan Tehranchi

June 16, 2021

## Introduction

Datava is a Software as a Service (SAAS) organization that focuses on developing CRM business tools for their clients; typically consulting firms and financial institutions. This includes providing enterprise-level management and business intelligence tools offered through a cloud-based platform. These business tools are provided through a suite of applications, providing users with software solutions focused on Customer Relationship Management (CRM), problem resolution, learning management, and predictive analytics.

One of the services Datava provides with their app is the ability for clients to view and manipulate their custom objects on Hubspot, a popular CRM tool that provides users with a suite of applications to help their business. The tools provided from Hubspot can utilize these objects to provide statistics and analytics information for the user to help them with their business and CRM needs. For example, an app on the Hubspot App Marketplace could interact with the user's “Clients” objects to provide statistics and information regarding their clients. These custom objects are vital to a user's experience on Hubspot, but currently, there is no easy front-end solution to manipulating these objects. The first part of our team's project solves this problem by creating a comprehensive user interface to build upon Datava's Hubspot manipulation software.

In addition to developing this interface, our team was tasked with implementing an OAuth protocol to connect Datava user's to their Hubspot accounts. OAuth is an internet protocol for securely accessing a user's information on another app without requiring their login credentials for that app. In this case, Datava's app is requesting a user's Hubspot information, so OAuth is needed to ensure that Datava can do this without handling the user's Hubspot credentials to do so. Accessing Hubspot's API with OAuth, rather than an API key, is a requirement for being listed on Hubspot's App Marketplace, which is something that Datava wants to do in the future. Implementing OAuth was the second part of our group's project, and a vital part to further improving Datava users' ability to interact and manipulate their custom objects on Hubspot. Through OAuth, users are able to safely login to their Hubspot accounts through a redirect on Datava's app; credentials and personal information are not stored within Datava's server.

## Requirements

There are numerous functional requirements for this project. On the OAuth side of things, we must successfully implement an OAuth integration between Datava and Hubspot. This means that an end-user should be prompted with an option to login to Hubspot on Datava's app, all through a secure OAuth connection. Additionally, the tokens associated with the OAuth login should be stored in Datava's database. This is necessary because OAuth's access tokens are set to expire every re-login or after a certain amount of time. By storing this information, Datava is able to prompt their users to reauthenticate their Hubspot connection inside the Datava app.

For the custom object handling, we need a way to create, modify, and delete Hubspot custom objects within the Datava app. These modifications must also be reflected in Datava's database. A data sync between Datava and Hubspot is another requirement set out by Datava. Information regarding a valid Hubspot custom object in Datava's database must be accurately reflected in Hubspot, and vice-versa. The user UI on Datava's end must be comprehensive and self-explanatory, whenever possible, requirements should be prompted through the front-end.

Non-functional requirements relate mostly to user privacy, security, and future-scalability. The integration of OAuth 2.0 is necessary for a safe and secure Hubspot login through Datava; this connection should not store any Hubspot credentials on Datava's database, like their username and password. For the software, post verification is necessary whenever the Datava app sends information for function calls. Post verification makes sure that the information sent to both Datava and Hubspot is secure, and adds additional protection for the database. Similarly, software variables should be flexible and not hard-coded. If an update to a variable is required in one file in the server, any additional dependencies should not have to be updated manually - allowing for future scalability.

Additionally, the code for this project, and any extra UI implementations should be well documented, with TODO statements left as necessary. Finally, as a future goal, Datava is looking to get listed on Hubspot's App Marketplace. For us, this translates to a successful OAuth 2.0 integration, as well as finding any information related to an App Marketplace listing.

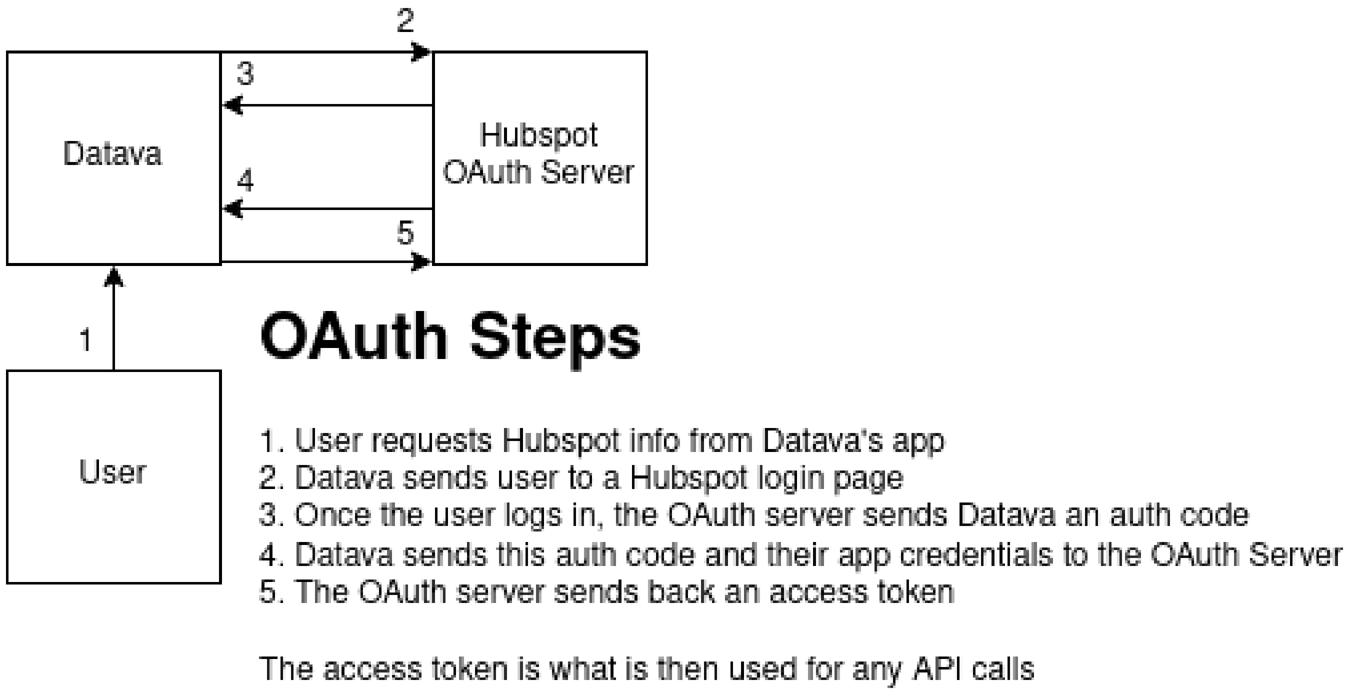
## System Architecture

### OAuth Implementation Design

The diagram in Figure 1 shows the current working flow for integrating OAuth with Hubspot on Datava's app. The process is broken down into five steps, beginning with the user instantiating the authentication from Datava's application and ending with Hubspot giving Datava a list of tokens, which can then be used in any API calls. This list consists of an access token, which is used to authenticate the calls, an expiration time, which states how long the access token is valid for, and a refresh token, which is used to get a new access token once it has expired.

This protocol ensures that Datava never handles the user's Hubspot credentials, leading to a secure connection between the user and their Hubspot objects through Datava's front end application. Another way we ensure security during this series of transactions is through ID's being sent between servers. Any request we send to Hubspot during this process is accompanied by an encrypted ID, which we check when the call is answered to make sure the response is from Hubspot and wasn't some attack from another server. Using OAuth, we can protect the user's information regarding their Hubspot credentials and information stored on their account, while still providing them easy, seamless access to their data on Hubspot.

**Fig. 1: OAuth Protocol Design**



**OAuth Steps**

1. User requests Hubspot info from Datava's app
2. Datava sends user to a Hubspot login page
3. Once the user logs in, the OAuth server sends Datava an auth code
4. Datava sends this auth code and their app credentials to the OAuth Server
5. The OAuth server sends back an access token

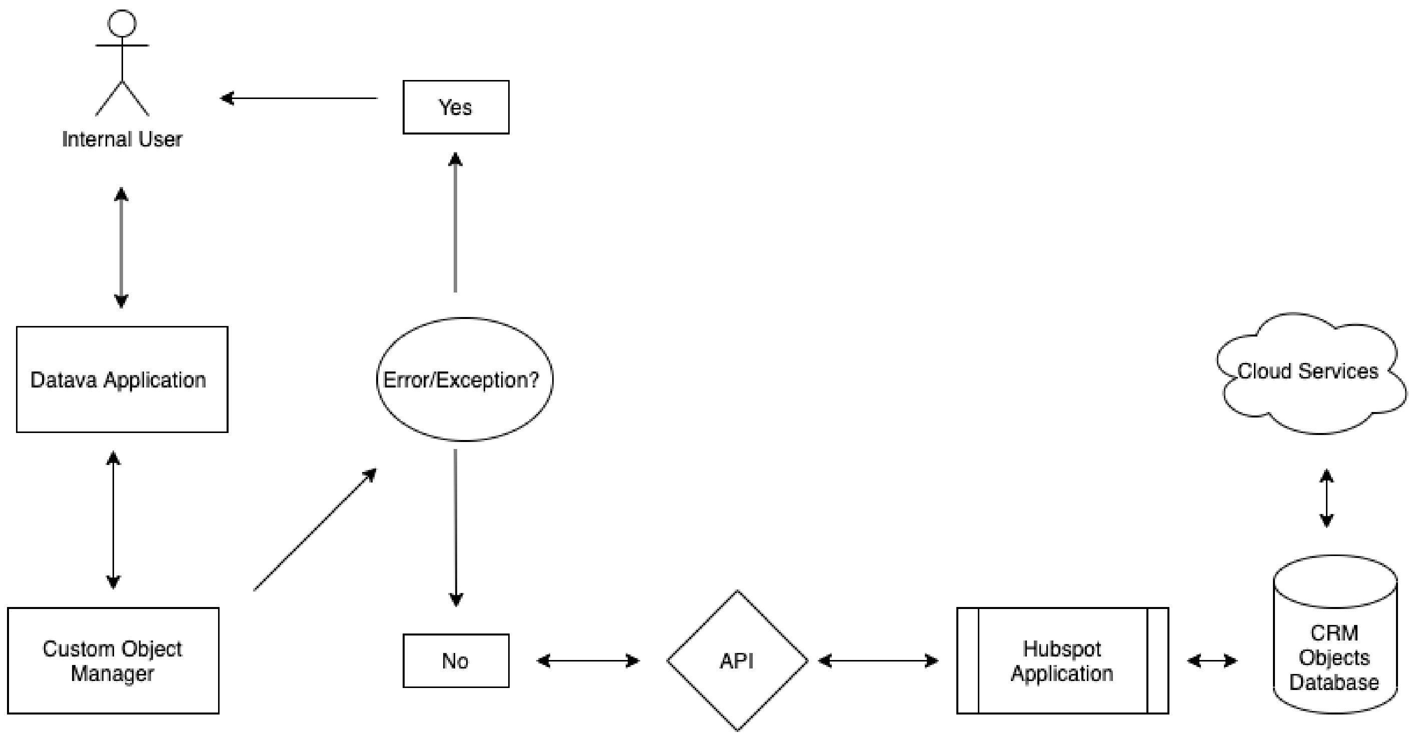
The access token is what is then used for any API calls

## Custom Object Handling and Design

As seen in Figure 2: Hubspot API UI and Object Design, internal users interact and manage their Hubspot objects through Datava's application. The interaction between users and the Datava application is back-and-forth, as Datava's UI directs users through the custom object process. Once the application has enough data, or the user attempts to create a custom object, a series of buttons and action listeners direct the data through what we called the "Custom Object Manager" (COM). The main function of the manager is to manipulate the data and prepare it for Hubspot. If there are any issues - e.g. missing fields - or any errors - e.g. the object already exists - the action is cancelled and a message is sent to the user. If there aren't any errors or exceptions, then the data goes through Hubspot's API. The API is responsible for creating, modifying, or deleting the custom object; these changes can be viewed through Hubspot's application as well as Datava's application.

For the sake of simplicity, we opted to leave some of the interactions out of Figure 2. For instance, the Hubspot application is able to speak directly with the Datava application, with Datava processing and handling responses from Hubspot in a way that internal users could understand. Additionally, the process of creating a custom object also creates a table entry in Datava's database. The creation of a Hubspot custom object requires that all necessary fields are filled out, while a table entry requires less of these fields. As a result, a valid table entry may be added while the Hubspot object creation itself fails. These scenarios are handled in the COM, returning a front-end response to notify the user.

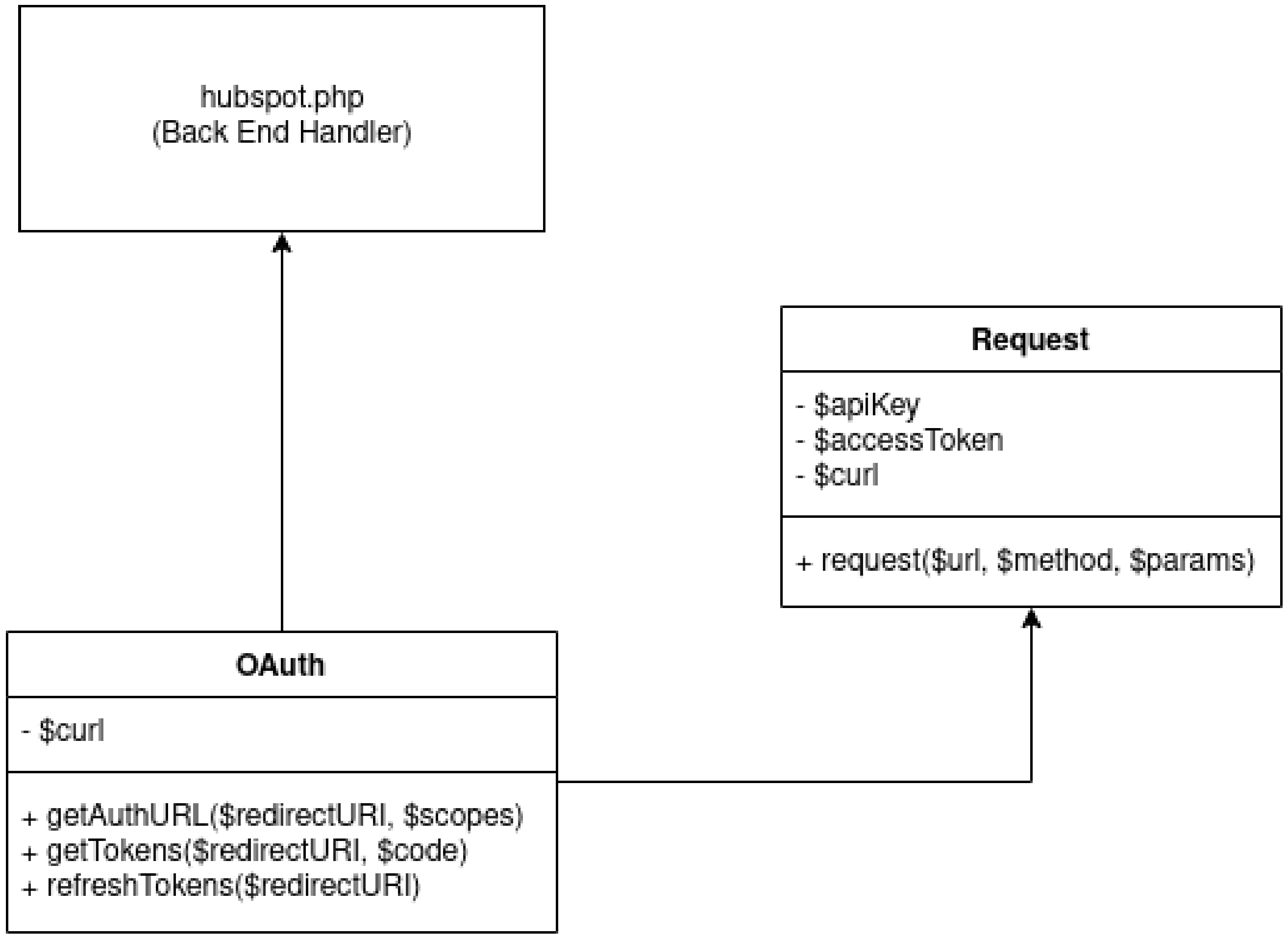
**Fig. 2: Hubspot API UI and Object Design:**



## Technical Design

Looking into our final product, it is divided into two key components: the OAuth 2.0 integration between Hubspot and Datava servers and the custom objects creation features in Hubspot. The OAuth 2.0 protocol first interacts with our product by having the end-user request Hubspot data from the Datava app. From here, our design then has Datava users sent to a Hubspot login page with the use of a redirect link (integrated function) for verification. After the end-user logs in using Datava-specific domain credentials, the OAuth server automatically sends Datava's server an authentication code to be used. At this point, our design has it set up so that once Datava has the authentication code, it will send that same code in addition to their app credentials to the OAuth server. Finally, the OAuth server takes in this data and verification and, if approved, sends back an access token which can then be used for all API calls Datava makes to Hubspot. If the end-user is not approved, however, the user credentials are revoked and the access token will be terminated. Figure 3 down below represents the backend of our OAuth Implementation.

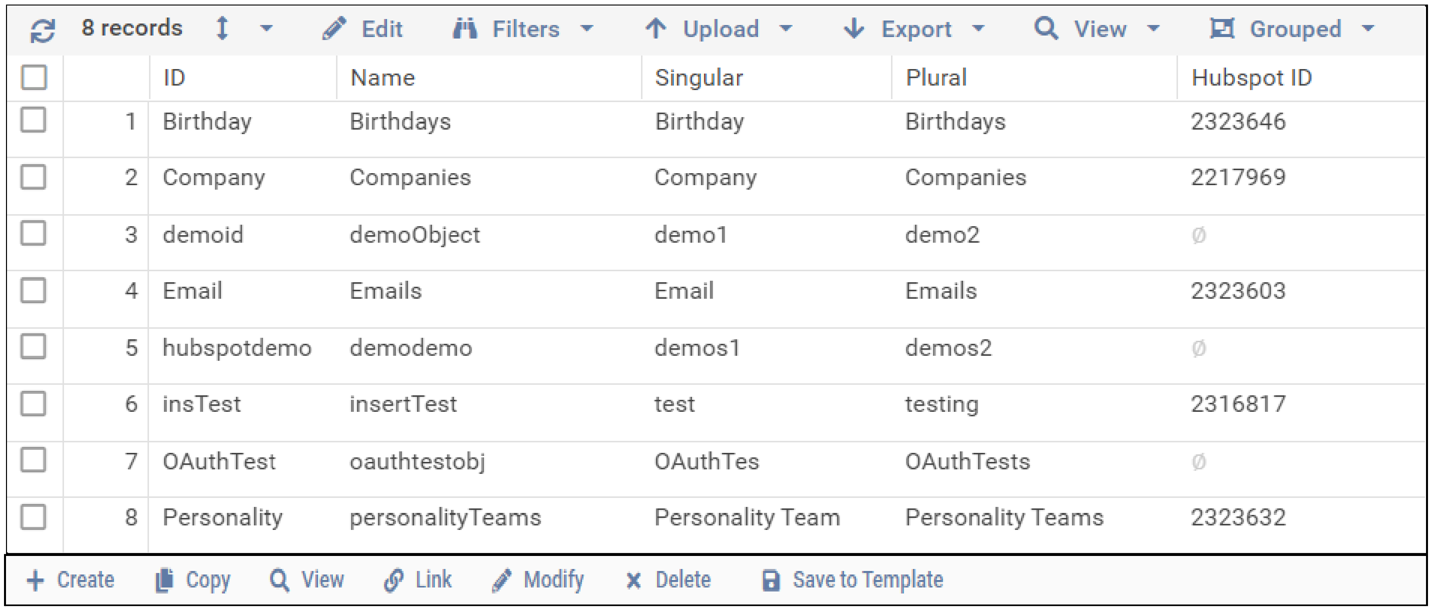
**Fig. 3: OAuth Implementation UML Design**


hubspot.php
(Back End Handler)
Request
- $apiKey
- $accessToken
- $curl
+ request($url, $method, $params)
OAuth
- $curl
+ getAuthURL($redirectURI, $scopes)
+ getTokens($redirectURI, $code)
+ refreshTokens($redirectURI)

When a user wants to create or manage their custom Hubspot objects with Datava, they have to first go to the Table Manager app on Datava's server. Next they have to go to the Hubspot Objects data table, seen in Figure 4 below, which lists all of the custom Hubspot objects that have already been made. From here, they can either modify an existing Hubspot object, or choose to create a new one.

With our implementation, opposed to the traditional objects that Hubspot would allow you to create such as company or email specifications, our custom objects button allows more in-depth objects to be created to suit the end-user's needs. A real world example of our Hubspot custom object implementation is within the CRM commercial field of Datava. This is where sales representatives can help classify their customers and clients based on needs and traits to ensure that the product delivery is optimized to them. In Datava's case, clients can receive optimized analytics and data for being able to kickstart their company in the right direction. Figure 4 below also represents an example case of the custom objects in use.

**Fig. 4. Hubspot Custom Object Database**



|  | ID | Name | Singular | Plural | Hubspot ID |
|---|---|---|---|---|---|
| 1 | Birthday | Birthdays | Birthday | Birthdays | 2323646 |
| 2 | Company | Companies | Company | Companies | 2217969 |
| 3 | demoid | demoObject | demo1 | demo2 | ∅ |
| 4 | Email | Emails | Email | Emails | 2323603 |
| 5 | hubspotdemo | demodemo | demos1 | demos2 | ∅ |
| 6 | insTest | insertTest | test | testing | 2316817 |
| 7 | OAuthTest | oauthtestobj | OAuthTes | OAuthTests | ∅ |
| 8 | Personality | personalityTeams | Personality Team | Personality Teams | 2323632 |

| QA - Software Quality Plan | |
|---|---|
| **Activity** | **Contribution to Quality** |
| Error Logging (PHP) | This activity allows us to find the source of front-end and back-end errors while trying to debug the Hubspot custom objects and UI. |
| PHP Unit Tests | This activity is extremely useful in allowing us to isolate certain functions and features in our Hubspot Integration and to be able to test that functionality specifically. |
| Jira (Atlassian) Scrum Tool | This activity not only keeps our team together through different user-stories and epics, however, also ensures that we are following our tasks based on the requirements. |
| Pair-Programming (QA) | This activity allows our team to be able to pair up with one other team member. Working as a pair compared to individually allows us to catch bugs, bounce possible thoughts off of each other more efficiently. Also, this activity ensures that our code is written more efficiently than individual coding. |
| Sandbox/Manual Testing | Using our local host, this activity allows us to emulate the source code onto our web browsers to be able to test our Hubspot app, UI, and OAuth functionality. |
| Visual Studio Code (VSC) Extensions | These integrations into Visual Studio Code help us as a team to be able to format, style and modify code more efficiently than a simple text editor. Furthermore, these extensions are known for helping users catch syntax errors much more quickly. |
| Discord (Audio-Chat) | This tool allows our team to be alert of code changes, meetings and enables real-time communication among our team. Discord also offers a screen-sharing option that allows our team members to validate each other's code and documentation. |

## Results

The goal of our project was to develop and enhance Hubspot's API, which Datava already has a baseline connection to. This has been completed through the means of implementing a robust GUI for accessing Hubspot objects and implementing source code to allow for the creation of custom objects. Furthermore, we secured end-user privacy through an OAuth 2.0 authentication integration where Datava domain developers can login and connect to their Hubspot account, through Datava's application. Our application meets all of our client's functional requirements, as it successfully authenticates users and sends token data to all verified Datava domain users (accomplished through Hubspot developer account testing) and enables custom object creation through our PHP GUI.

The integration of OAuth 2.0 was done entirely through a back-end environment (PHP) and testing varied throughout the development process. There were many milestones where we could send front-end notifications to end-users, such as a successful login screen, or a domain redirect. When we were not able to verify via front-end, we redirected variables through console or debugger logs. This allowed us to see what we were working with, what we were expecting, and what we might need to do. Testing of the finished integration was mostly straightforward. A successful connection begins with a prompted connection screen, a few redirects, and ends with a redirect back to Datava's app. Additionally, we were able to verify a successful OAuth login by creating and modifying a Hubspot custom object through Datava's app after the OAuth login.

The buttons were made with multiple function calls and figuring out what to pass in to functions, as a result testing revolved around console and debugger logging, as well as some UI prints to the user. Once the individual features were tested, component testing was utilized. For component testing, we set up some test objects and then focused on creating, modifying, and deleting them. We were able to check the objects table in Datava's database to verify object creation on their end, as well as checking our Hubspot accounts to verify custom object creation. A similar process was used to verify that the action listeners worked as intended. However, the action listeners were set up in a way to return UI messages to end-users, so their tests revolved around verifying correct errors/responses for different inputs and actions.

## Future Work

Additional features could be added to provide new end-user capabilities down the road. Throughout the project, extra features - or reach goals - have been requested by Datava. As the implementation of these features was described as “fairly straightforward,” we chose to focus on implementing necessary functions, such as object managing buttons and action listeners. A few buttons have been left out and could easily be added in future work. The button code is pretty similar across all buttons, and their addition could be accomplished by examining existing code and documentation left by our group. Other features, like a way to create associations between objects, could be added with relative ease, as Hubspot has extensive documentation on their CRM features.

The button and action listeners which we created were focused on just one table in Datava’s database, which manages the custom objects. However, in the current implementation, it is not possible to create a custom object with one single action. Instead, you must first make an entry for the custom object table, and then make a custom object property in a separate table before you can create the actual object. For future work, a similar table managing component could be added to the properties table, where the creation of a primary display property could automatically update the object table entry and create a custom object on Hubspot. The addition of action listeners to the property table wouldn’t be too much work, but it would require some additional query calls to update the objects table. However, it might be easier to simply add a way to create an object property while creating the object entry itself. This would require some changes to Datava’s UI, as they would need some extra fields for inputs and additional rerouting. Either way, the creation of Secondary Display Properties would need to be taken into account, as these are also made externally before being added to the object table and pushed to the corresponding Hubspot custom object.

With our project conclusion, there are direct buttons for creating, getting, modifying, and deleting Hubspot custom objects, and there are action listeners which do the same thing through the use of the submit button. The creation of these two components stemmed from a reach goal, where Datava sought to replace the button functionality through the use of action listeners. As we had limited time for the project, both components were developed, though Datava can replace or remove the button component entirely down the line. The automation which comes with action listeners means that ideally, only the “delete” button should remain, as all other functions can be checked in the after insert and after update listeners. Replacing or removing three or four of these buttons would be as simple as removing some Javascript code on Datava’s server.

For future developers, we left documentation of our work, following comment examples from Datava’s own source code. We also wrote additional comments wherever we deemed necessary. In the php code, we left TODO statements where our client added reach goals. In the Javascript code, we documented function flows and left comments describing desired functionality for any unimplemented features.

## Appendix

### I. Common Terminology

- API - API is an acronym for Application Programming Interface, this software intermediary allows two applications to talk to each other. In our case, Datava's application is able to talk to Hubspot.

- CRM - Customer Relationship Management, CRM is a technology that is used for managing your company's relationships and interactions with existing and potential customers. The goal of CRM is to improve business relationships to help grow your business.

- Hubspot Custom Object - by default, Hubspot offers four CRM objects for their users: contacts, companies, deals, and tickets. Hubspot also gives users the ability to define and create up to 10 of their own CRM custom objects, with features such as name, singular and plural forms, and display properties for the object.

- Javascript - an object-oriented computer programming language which is commonly used to create interactive effects within web browsers.

- PHP - PHP is a recursive acronym for Hypertext Preprocessor. It is a widely-used open source general-purpose scripting language that is suited for web development and can be embedded into HTML.

- VSC - Visual Studio Code, VSC is a code editor focused on building and debugging modern web and cloud applications.