CptS 422 Project: Deliverable 1

BATS: Business Analytics Tracking Service

Austin Marino, Nicholas Kent, Joseph Cunningham, Cole Bennett

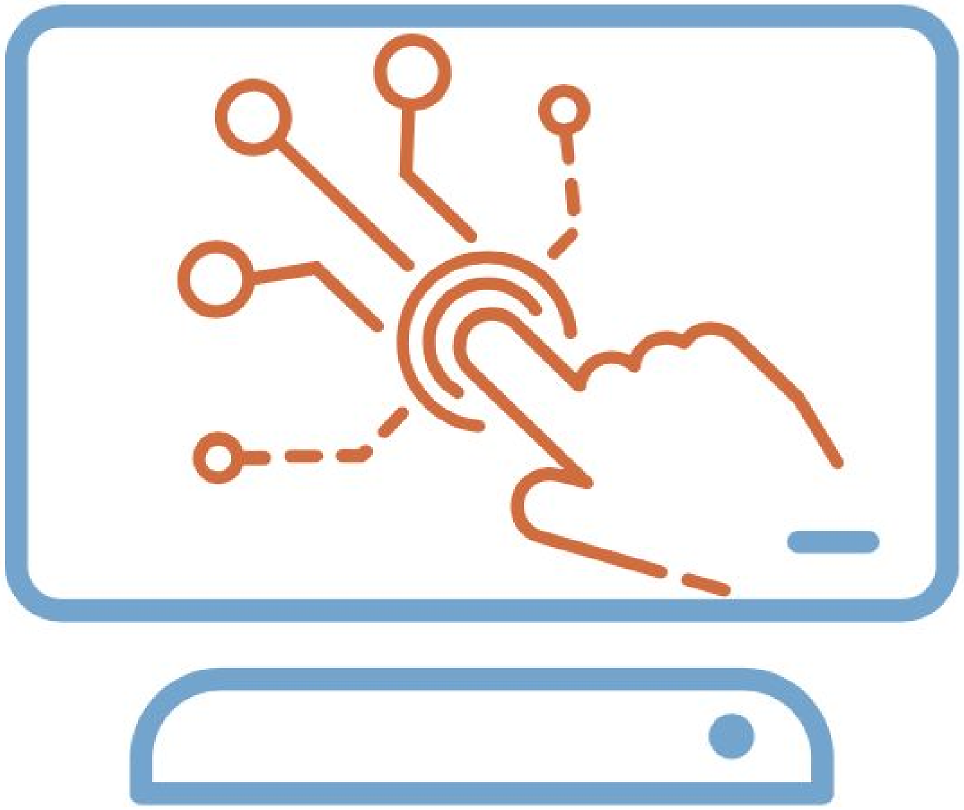


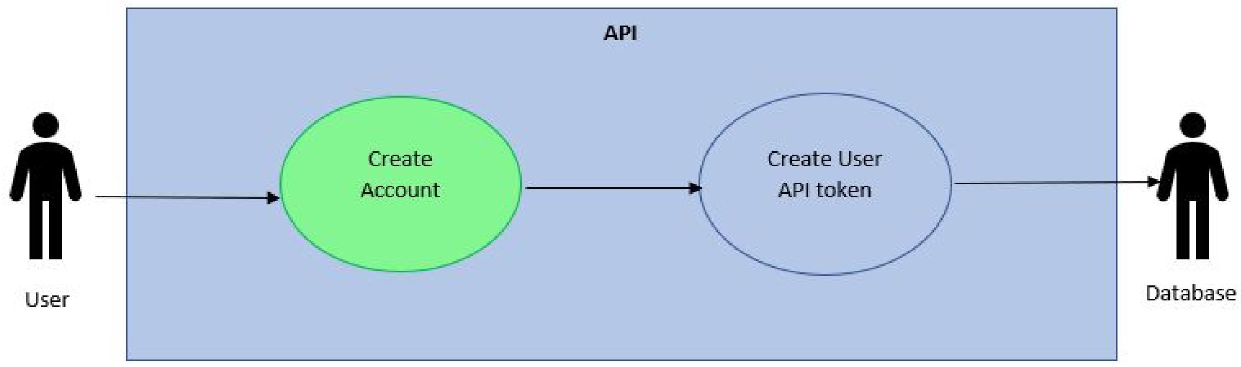
Table of Contents

Requirements Specifications

Use-Case Diagrams

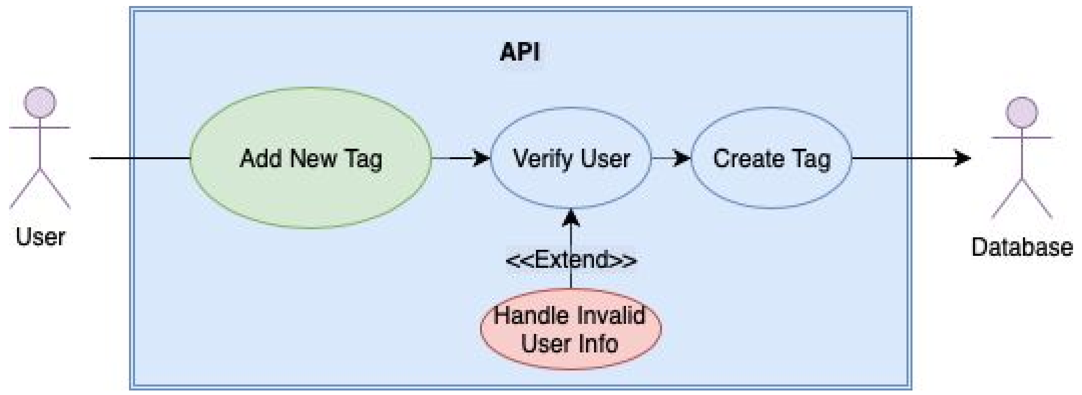
User Creates an Account

In this scenario a user wants to create an account and thus needs an API token to be created in our database to be able to identify and store data for that user in the future.

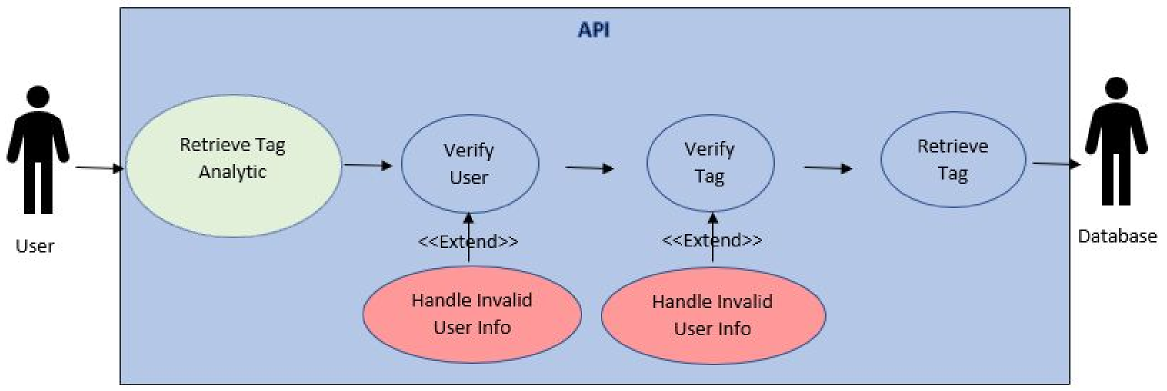


User Creates a Tag

In this scenario, a user wants to create a new tag within our database.

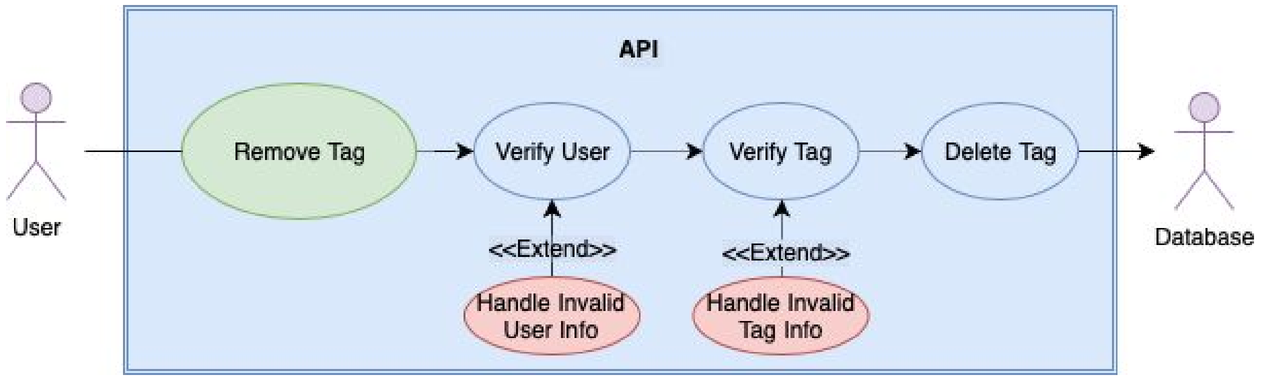


User Retrieves Tag Analytic



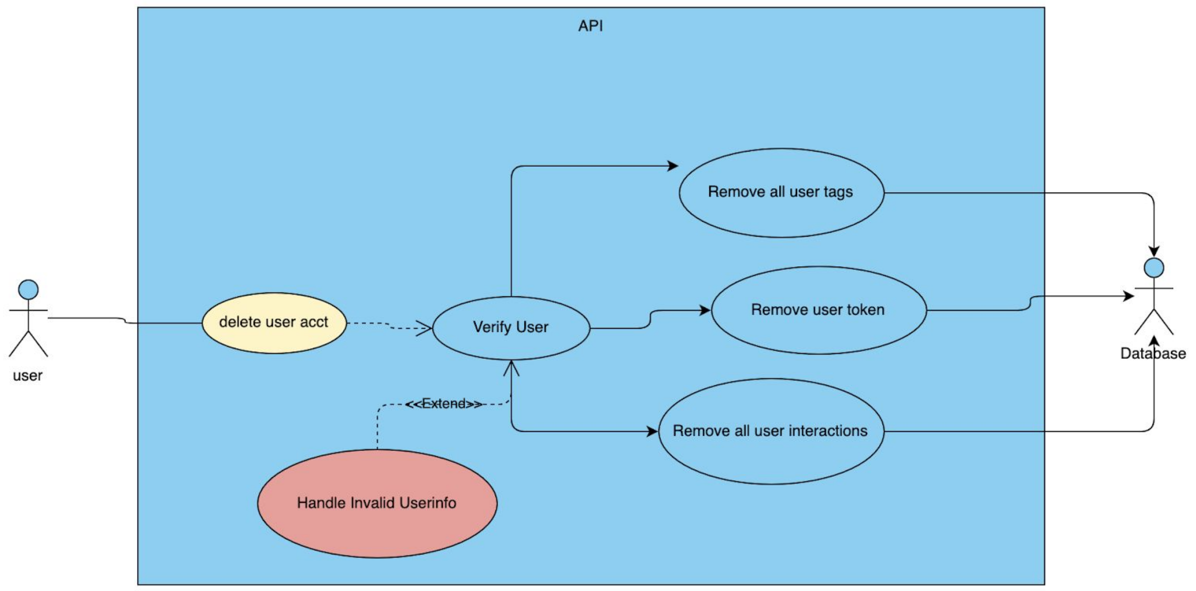
User Removes a Tag

In this scenario, a user wants to remove an existing tag that they had previously created within our database.

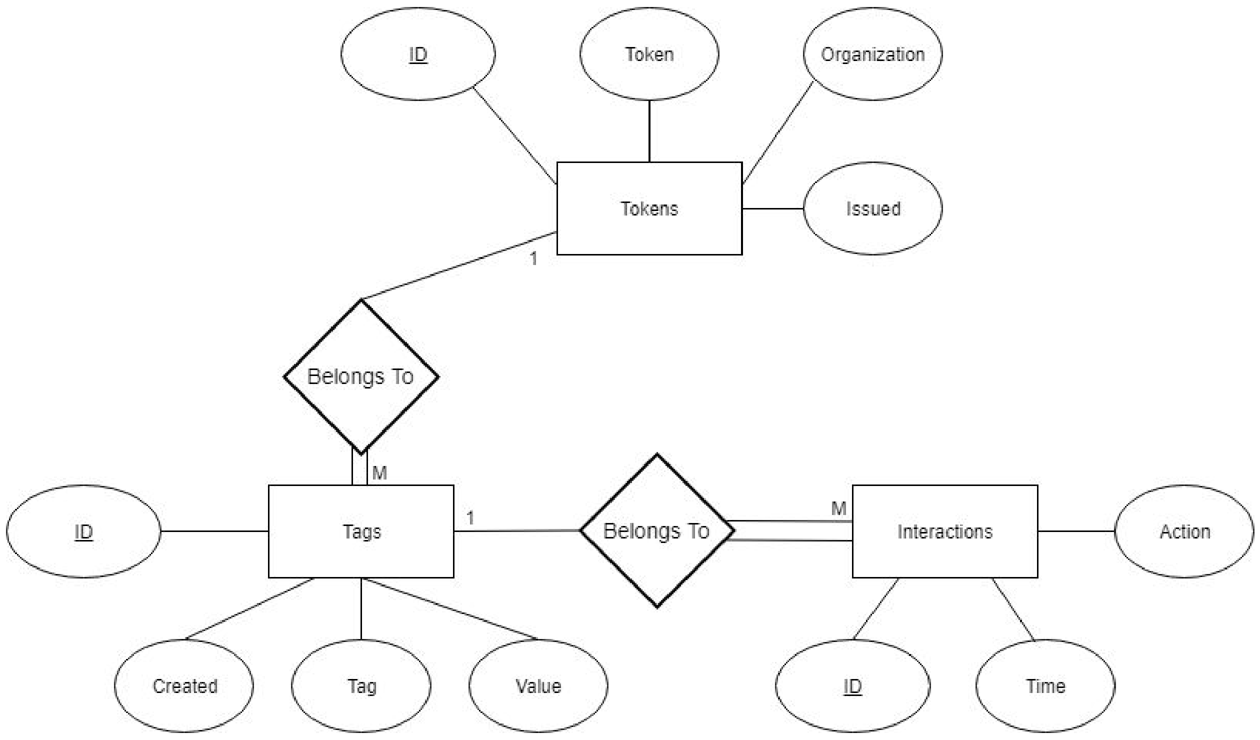


Delete a User's Account

In this scenario, a user wants to delete their account and thus wants to remove their stored data within our database.



Class Diagrams



(Entity relationship diagram)

The backing store for our service will utilize a relational database, specifically MySQL.
We will compose our model into the following relations:

- **Tokens**: Contains all information pertaining to tokens.

id	Serial integer	Primary key
token	VARCHAR	UUID of the token
organization	TEXT	Name of the organization which requested the creation of the token
issued	TIMESTAMP	Date and time of creation

- **Tags**: Contains all information pertaining to tags.

id	Serial integer	Primary key
token_id	Foreign key	Reference to ID of Tokens
tag	TEXT	Name of the tag with period delimiters.
value	TEXT	Tag metadata
created	TIMESTAMP	Date and time of creation

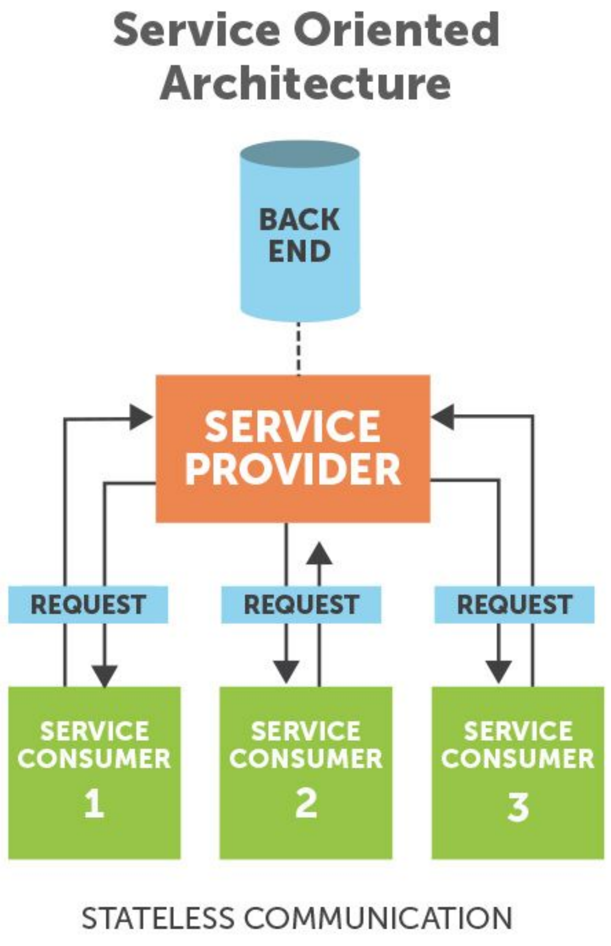
- **Interactions**: Contains entries for each interaction associated with a specific tag.

id	Serial integer	Primary key
tag_id	Foreign key	Reference to the ID of Tags
action	TEXT	Metadata to denote the type of interaction (i.e. Button Click)
time	TIMESTAMP	Date and time of interaction

Design Specifications

Architectural Context Diagram

For our project, we decided the Service-oriented architecture (SOA) since our end product is an API that provides our services to other components through a communication protocol. The basic principles of service-oriented architecture are independent of vendors, products, and technologies; which, means that that the services we provide are self-contained and act as a black box for our consumers. Because of this flexibility and potential for reuse, many different client types can use our service. Additionally, SOA evolved from the design principles of Object-Oriented languages and thus made it an easy adaption for our team to understand and implement. Overall, SOA allows us to translate design principles to an Enterprise level system and establishes a reusable library of loosely coupled services for our customers.



Detailed Component Diagrams

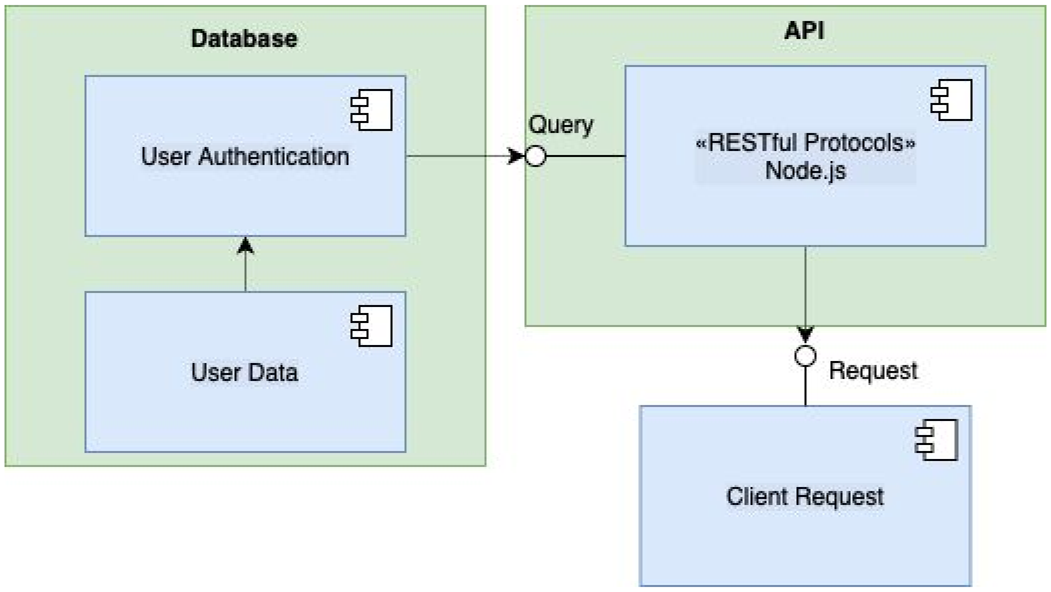
Database

Our database consists of three essential tables. The first table invoked during any query is the “Tokens” table. This table contains all the API tokens assigned to each unique user of our system. Thus anytime a user tries to interact with our system, they need to provide their API key to get access to their data. If they do not have a key already assigned to them, the user can request for one to be created for them. Once a user has a key, they can use it to access their tag data stored in the “Tags” and “Interactions” tables.

API

This section of our component diagram contains the code that the user communicates with when they send a request to our server. Our codebase is written in Node.js and follows the RESTful API protocol for all user requests. When a user sends a request to our server, our code processes it and submits the request in the form of a SQL query to our database.

Displayed below is a component level design of our architecture system.



Code Implementation

Codebase Url

https://github.com/austinmm/WSU_CptS_422_BATS

Version Control System

Using GitHub for version control and setting up the CI/CD Pipeline.

Languages Used

Frontend

- **Languages**: Html/CSS Javascript
- **Reasoning**: This will be necessary for simple demo purposes of how anyone could leverage our analytics API and setup proper tagging for components on their frontend.

Backend

- **Languages**: Node.JS
- **Reasoning**: Node JavaScript will be the language used to build and deploy our server. Node makes running a server with simple tasks extremely easy and will help us speed up development so we can focus on features and proper design.

Frameworks Used

Express

Express is the most common framework users use with Node JS. It further simplifies creating a server instance, managing it, and deploying it.

MYSQL

The MySQL framework simplifies the process of connecting to a SQL database and querying it.

Other Development Tools Used

SQL Database

We created a relational database on hosted on Azure due to ease of use and getting slightly more sophisticated information based off of our data without needing to setup a local db.

Postman

Postman is a great tool for manual testing. It allowed us to easily invoke our endpoints and also has the potential to run automated test scripts for future deliverables.

IDE

There are many capable IDEs to choose from but we mostly used Visual Studio Code to due its many extensions/add-ons and developer friendly environment.

Software Documentation

Documentation of the project will be present in the following ways:

1. All functions, classes, and variables of importance will be associated with concise and descriptive comments, formatted according to the industry-standard conventions of JavaScript.
2. Markdown and PDF files will be utilized as the formats for documentation files in the project repository. The following documents will be included:
    a. A readme document containing general information and installation/build steps.
    b. API document to describe in detail all API endpoints that our service will provide. Information regarding request methods, request formats, status codes, and response formats will be described in detail.

Milestone Report

Joseph Cunningham

During the first deliverable, I designed the demo frontend application in html, css, and javascript. The application is just a simple html document that is designed to showcase our API as it makes successful http post requests using ajax once you click one of the four available buttons on the screen. I also wrote the software process section of this document and was responsible for two of the use-case diagrams, the create account and retrieve taga analytic, which I created using shapes and icons in a word document.

Cole Bennett

For deliverable 1, I worked on the backend design for our application. I created the class diagrams, which include the entity-relationship diagram. I also wrote the details describing our schema for our application’s SQL backend. Secondly, I wrote the Software Documentation section of this document, which includes details on how we will document our code and project.

In addition, I created the SQL statements to create the tables, and the mock script which populates our database with data for testing. This also required creating a database library class which is utilized by the endpoints.

I also created the documentation file which is an OpenAPI specification that describes our endpoints. This provides us with a way to update our specification in version control, and also lets us view the endpoints in an interactive web page (swagger.io). In the next deliverable, we will write our unit tests against this specification.

Nicholas Kent

For deliverable 1, I worked on setting up the foundation for the server and writing a simple setup that would prepare Node js to be run on MacOS or Linux system and install all dependencies. This included mentoring and overlooking the team as well as creating examples for the team to work off of.

Created a library for simple query interactions working asynchronously with sql server within nodejs. Wrote boilerplate to demonstrate how this small library could be used and paired with any sql server given the proper connection information.

Created a script for launching the node server and pairing it with website for easy testing and interactions. Updated the README.md to reflect simple steps to get the build running and have it be easily testable.

Austin Marino

For deliverable 1, I worked on numerous parts of our documentation and development process. While doing this, I also served as our team liaison, arranging meetings, assigning tasks, and overall ensuring our team was staying on task and schedule.

The central part I worked on, in this document, was the design specifications section. I wrote about the design architecture our team was going to use for the implementation of our codebase. I also created the component diagrams which depicted how we would implement the service-oriented architecture model we are using. In doing this, I had to research what software architectures were best suited for implementing an API and in turn, learned about SOA. Additionally, I created the use-case diagrams for the scenario of “User Creates a Tag” and “User Removes a Tag”.

When it came to implementing our codebase I created most of the endpoints for our API. I worked specifically on endpoints…

- GET ‘api/tags/<name:optional>’
    - This endpoint returns an organization’s tags specified by the name paramaterer. If the name parameter is left empty, not specified, then it returns all of an organization’s tags.
- POST ‘api/tokens/<organization>’
    - This endpoint creates a database entity inside our tokens db for the organization specified by the parameter. While creating the db entry it also generates a unique

token for the organization so they can authenticate themselves in future API calls.

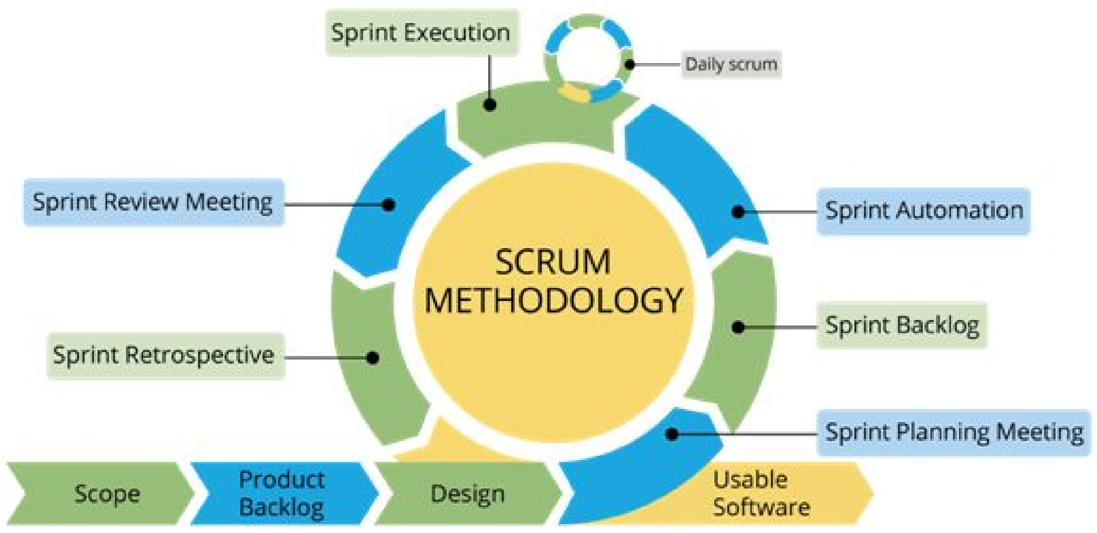
- POST ‘api/tags/<name>?interaction=<interaction type>
    - This endpoint creates a new tag, if it doesn’t already exist, for the authenticated organization and then creates and assigns a new interaction db entry to the tag specified.

Besides creating these endpoints I also made small changes to our codebase to improve certain features. These changes include updating the name column in our tags database table to make it a unique column so there could never be duplicate tag names. I also created a catch all middleware route which checked if an API call was made with the correct Bearer authentication and if it was if the token was valid and belonged to an organization in our tokens database table.

Software Process

For this project, we’ve decided to utilize the Scrum methodology. Scrum is a type of agile process which means that this method is suitable for small projects such as this one and is also very flexible, allowing for the requirements to change if they need to. It’s easy to understand and facilitates team collaboration and organization as we will work in week sprints and evaluate at the end of each one.

So far, for deliverable one we have only had one sprint which was a week long and consisted of us coding the api with a few endpoints along with the demo frontend application. Before the sprint started we had a meeting to organize it and divide tasks equally among the members of our team. At the end of the sprint we had another meeting for discussion and review before our project was submitted. For future deliverables we will continue to stick to the scrum methodology mold of planning, sprint, and review, as well as other key parts of it.



Project Activities

Meetings

We held weekly meetings every Monday from 3-4pm.

8/26/19: Discussed and ranked our project ideas.
9/2/19: Picked our final project idea to go forward with.
9/9/19: Worked on the design of our system (including the database and endpoints).
9/16/19: Implemented our deliverable 1 requirements.
9/22/19: Final meeting for the deliverable 1 submission (general cleanup).

Our team conducted weekly meetings where we discussed current issues that needed to be addressed and any future work that was required. During this time, we also conducted code reviews and assigned a future task for each member of the team to ensure that everyone was doing an equal share of the work. We documented who was assigned what tasked in our Slack channel which we regularly communicated on a nearly daily basis. Please visit here to view our messages on Slack.

Problem/Solutions

Some of the major challenges we faced in this deliverable were new software tools and languages that were unknown to all or some of the team before we began the project. Frameworks like Node.js and Express provided an initial challenge since not everyone had experience using them and thus required research and tutorials before any coding could get completed.

We also wanted to create an API specification document so that users could easily understand how to navigate our endpoints. To do this Cole researched different documentation tools and formats until he ultimately decided we should use OpenAPI standards to write a YAML file which can be uploaded to editor.swagger.io for easy visualization and understanding of the structure of our API.