Group Event Planning App

Recommending Events For You

Creative Project
Barrett, the Honors College

By:
Preston Russell and Connor Sonnier

Director:
Dr. Yinong Chen

Secondary Reader:
Dr. Ryan Meuth

Table of Contents

Abstract

For our creative project, we initially wanted to work on a web application that would allow people with busy schedules to easily create and share events while also discovering other events that may interest them. With that in mind, we created the Group Event Planner App, a full stack project that lays down a foundation for all of our goals while focusing primarily on the proposed recommendation algorithms that enable its users to discover events that are likely to pique their interest. The development of our recommendation algorithms took inspiration from existing implementations, such as those at Amazon, YouTube, and Netflix, and resulted in a creative amalgamation.

Objectives

In order to help people decide what to do with their free time easily, we wanted to create a web application that would allow users to subscribe to events they are interested in and then receive recommendations for other events that they might be interested in. In order to achieve this, we set up the following objectives to drive our project:

- Develop a recommendation algorithm to suggest events to users based on what they have already subscribed to
- Develop a recommendation algorithm to suggest events for each event based on the details of the event itself and what other users who have subscribed to the event have also subscribed to
- Create a web service that allows us to interact with the database to create, modify, and retrieve events, categories, and users

- Build endpoints within the web service to orchestrate the creation of random data for users and events on a very large scale in order to test the algorithm with a lot of variety
- Create a responsive web application to easily use our web service and demonstrate the recommendation algorithms from the point of view of a potential user
- Build the functionality within the web application to impersonate users and add events from their perspectives in order to see how the algorithms behave

Literature Review

CSE 446

Arizona State University's Dr. Yinong Chen instructs a course titled "CSE 446: Software Integration and Engineering", in which one of the topics that he discusses is recommendation algorithms. He defines a recommendation system as one that "ranks a set of available choices based on certain given criteria and available set of inputs" and "predicts the rating, ranking, and preference, which a user would give to an item". In his lecture, Dr. Chen discusses the value of caching information such as "Most Frequently Bought" and "Frequently Bought Together" items and the possibility of using this cached information as input factors for recommendation algorithms. Using Amazon.com as an example, Dr. Chen explains that "Most Frequently Bought" and "Frequently Bought Together" items are, as the names suggest, the most popular items and items that users often purchase in the same order, respectively. Since this data is constantly changing as users buy more items, it is worth caching to save resources and to increase the speed of the recommendation system. Dr. Chen also provides an overview of some recommendation strategies in his course. These are categorized as content-based filtering, user profile filtering, situation-aware filtering, and collaborative filtering. Content-based filtering filters the available choices based on "the attributes and semantics of the concerned

items". User profile filtering filters the available choices based on the attributes and behavior of the user, situation-aware filtering filters the available choices based on the company's situation, and collaborative filtering filters the available choices based on how users collectively interact with them (Chen 2020).

Our event planning application's recommendation algorithms are similar to those which Dr. Chen discusses in his course, as they rank the available events based on a set of given factors and how they should be weighted for the purpose of predicting how likely the user is to attend the event. We could have used the "Most Frequently Bought" items, or in the context of our application, the popularity of events as a factor in our algorithm, but decided not to because an event's crowdedness is not an indicator of how enjoyable it would be to attend. However, we did implement a factor similar to "Frequently Bought Together" items. In our event planning application, this would correspond to events that have been attended to by the same user. If one hundred users have attended both event A and event B, then users who have attended event A are more likely to enjoy event B than users who have not attended event A. In these ways, our event planning application's recommendation system takes advantage of content-based filtering, user profile filtering, and collaborative filtering.

Amazon

Amazon is the world's largest eCommerce platform, offering anything one might want to buy from food to computer components. With so many items to choose from, providing customers with high-quality recommendations can greatly increase Amazon's revenue. With this in mind, Amazon created a recommendation algorithm called 'item-to-item collaborative filtering', based on the popular collaborative filtering algorithm. The normal collaborative filtering algorithm matches the user to similar customers to provide recommendations to the user. Item-to-item collaborative filtering matches each of the user's purchased and rated items to

similar items, then combines those similar items into a recommendation list. Amazon's item-to-item collaborative filtering provides high-quality recommendations regardless of how much user data they have because this algorithm focuses mostly on the data that Amazon has for their items, not users. However, the scalability and performance of this algorithm is dependent on the automated offline generation of the similar-items table, as this table's creation becomes a very expensive operation at larger scales in terms of storage and processing power (Linden 2003).

For our event planning application, we will not be collecting very much user data, but instead we will be focusing on events. This is similar to how Amazon focuses on the items that they sell instead of collecting a large amount of user data. Due to this similarity between the context of our application and the context of Amazon, item-to-item collaborative filtering is a suitable option for our recommendation algorithm, but the need to create the similar-items table offline makes this algorithm less dynamic than we would like.

YouTube

YouTube is an extremely popular video-sharing platform with millions of videos in their database. With "one billion hours watched daily" and "2+ billion users" ("YouTube for Press"), there is a lot of data that YouTube needs to comb through to provide a user with video recommendations. Due to YouTube's gigantic scale, incredibly dynamic content, and noisy training data, YouTube had to get creative with their machine learning-based recommendation algorithm. Usually, machine learning-based recommendation algorithms consist of a single neural network that ranks the options and then recommends the highest-ranked options. YouTube additionally implemented a candidate generation neural network that takes events from the user's YouTube activity history, selects a few hundred videos from the millions in YouTube's database, and then utilizes collaborative filtering to provide a personalized list of a

few hundred recommendations. Then, YouTube's ranking neural network takes these few hundred video recommendations, assigns a score to each video using both features of the video and of the user. The best-scored videos are then recommended to the user (Covington 2016).

Since our event planning application's content isn't so dynamic and its scale isn't as large as YouTube's, this algorithm is over-the-top for our purposes. However, the idea of generating candidates for recommendations before ranking them could be utilized to fix the problem of Amazon's item-to-item collaborative filtering requiring that the similar-items table be created offline.

Netflix

Netflix is one of the most popular platforms for streaming movies and shows. As Netflix's catalog of movies and shows are only in the thousands, as opposed to the millions, Netflix didn't need to focus so much on efficiency with their recommendation algorithms and could instead dedicate the majority of their resources to developing many sets of personalized, high-quality, thoroughly-tested recommendations. The Netflix homepage consists of "about 40 rows" (Gomez-Uribe & Hunt 2016) of recommendations, each row generated by a different recommendation algorithm. Some rows give recommendations based on genres, some based on particular movies or shows that the user has already watched, some based on similar users, etc. There is also another recommendation algorithm in place that decides which rows to put on the user's homepage, and in what order (Gomez-Uribe & Hunt 2016).

Our applications users are allowed to create events, so our recommendation algorithms need to be prepared for data on a larger scale than Netflix's. Therefore, we cannot approach our recommendation system in the same way that Netflix approached theirs. However, it would be beneficial to take inspiration from some of the factors that Netflix bases its many recommendation rows on.

SeatGeek

SeatGeek is an aggregator and search engine for event tickets that tries to inform its users about how good of a deal each ticket is. To do this, they developed their recommendation algorithm based on a “Deal Score”, which takes several factors into account including seat quality, historical ticket prices for the event or performer, the popularity of the event, the quality of the currently available tickets, and the prices of the currently available tickets. This algorithm is meant to represent a fair market value for the ticket and compares this value to the current price of the ticket to generate the “Deal Score”. This makes customers feel more confident in their purchases, increasing the company’s revenue (IR).

SeatGeek’s “Deal Score” is similar to the fitness value that we are proposing in our algorithms, as these are both values that are assigned to every event/ticket that we display to our users based on the objectives of our services. However, SeatGeek’s “Deal Score” algorithm seems almost entirely based upon content filtering, while our proposed algorithms also utilize user profile and collaborative filtering.

Eventbrite

Eventbrite has been assigned a patent that describes an event recommendation algorithm that utilizes social network information, event history information, and event information. In other words, this algorithm is based on collaborative, user profile, and content filtering. Based on the social network information and the event history information, this algorithm scores a “friend connection” between every pair of two users and creates a tree of users and their connections to the user that the algorithm is recommending events to. This connection is based on social network information associated with the two users and event history information associated with the two users. Based on this “friend connection” tree and the

events' information, the event listings are ranked (Zambrano & Groesbeck "Social event recommendations").

Our proposed recommendation algorithms also use a form of collaborative filtering, but they would benefit from a friends system. Our algorithms would be able to use both their current form of collaborative filtering as well as a "friend connection" tree like the one described in this patent. This would increase the accuracy of our algorithms' fitness scores, which would improve the recommendations that they generate.

Microsoft

Microsoft has been assigned a patent for an event recommendation service that aims to recommend events that the user is likely to attend and that are near the location of the user by utilizing selection data, location data, event data, and social networking data. Selection data refers to media content selected by the user. Selection data, location data, and event data are used to generate a recommendation for an event that the user is likely to attend and that is near the user's location and then a list of people who are connected to the user and are likely to attend the event is generated. The recommendation and the list of people are then communicated to the user. Then, a person who is known to the user, likely to attend the event, and has not been invited to the event is selected and an invitation to the event is communicated to this person (Murphy, Jensen, Weare, Evans, & Gibson "Event recommendation service").

The goals of the algorithm described in this patent and the algorithms that we are proposing are the same: to recommend events that the user is likely to be interested in and attend. Since they have the same goals, these algorithms also have some similarities in functionality. Our proposed algorithms heavily rely on selection data (media content selected by the user), where the media content correlates to events. They also both utilize location and event data.

Fuzzy Relational Approach

Chris Cornelis and his research group at Ghent University have developed a recommendation algorithm that utilizes collaborative filtering and content-based filtering to model user and item similarities as fuzzy relations. They summarize their approach as "recommending future items if they are similar to past ones that similar users have liked". This approach emphasizes the fact that events should not be recommended after they have already taken place, so recommending events based on ratings does not make sense. The algorithm is based on the idea of modeling user and item similarities as fuzzy relations, which essentially means that users do not either prefer nor not prefer an item, but instead a user's preference for an item lies within a range from 'does not prefer' to 'does prefer' (Cornelis, Guo, Lu, & Zhang 2005).

Some aspects of our proposed recommendation algorithms are very similar to the algorithm that Chris Cornelis and his research group have proposed. Our algorithms also realize the fact that events should not be recommended after they have already taken place. This is why we did not see much value in implementing a rating system, as an event should only be rated after it is attended. Also, the fitness scores that we assign to each event in our recommendation algorithms is similar to the fuzzy relations that Chris Cornelis and his group use to model user and item similarities, as both represent a user's preference for an item as a value within a range.

Ticketmaster

Zhen Qin's group at Ticketmaster has developed an approach for email recommendations that is scalable and effective. They added variation to the results of their recommendation algorithm in comparison to collaborative filtering-based algorithms because it

would be ineffective to send the same email week after week. To make their machine learning algorithm scalable without sacrificing its effectiveness, Zhen Qin's group implemented several machine learning methods such as using online bootstrapping as a heuristic for their contextual bandits, feature hashing, and learning reduction techniques. Their resulting algorithm trains on billions of features within minutes on a personal computer.

Our recommendation algorithms are meant to be online and cannot afford to take minutes, so we were not able to implement any of the techniques used for Ticketmaster's periodical email recommendations. However, our algorithms may have been able to utilize some machine learning strategies to become more scalable.

The Proposed Recommendation Algorithms

For Each User

This recommendation algorithm takes an integer *n* and a user as input and then generates *n* recommendations based on the user's data and the data of the events that the user has subscribed to. This algorithm is a mix of collaborative filtering and item-to-item collaborative filtering. To generate the recommendations, a fitness score is generated for every event and then the events are ranked based on this fitness score. The highest-ranking events are the ones displayed to the user. The fitness score is intended to represent how much the user would be interested in an event and is generated based on five weighted factors:

i	Factor	Weight (fitness score)
0	The percentage of keywords in subscribed events' titles that also appear in this event's title	15 * this percentage
1	The percentage of events that the user has subscribed to in the same category as this event	15 * this percentage
2	Whether this event is in the user's state	10
3	The percentage of the user's subscribed events that are in this event's state	10 * this percentage
4	The percentage of similar users that have subscribed to this event	The sum of all other factors * this percentage

Table 1: Fitness Score Factors for 'For Each User' Algorithm

Similar users are defined as users who have subscribed to events that the input user has also subscribed to. The fitness score *F* for each event, where F_i is the score contributed by factor i, is calculated as:

$$F = \sum_{i=0}^{4} F_i$$

$$F_0 = 15 * \frac{\#\ of\ matching\ keywords\ in\ this\ event's\ title}{\#\ of\ keywords\ in\ this\ event's\ title}$$

$$F_1 = 15 * \frac{\#\ of\ subscribed\ events\ in\ this\ event's\ category}{\#\ of\ total\ subscribed\ events}$$

$$F_2 = 10 * (1\ if\ this\ event\ is\ in\ the\ user's\ state,\ 0\ if\ not)$$

$$F_3 = 10 * \frac{\#\ of\ subscribed\ events\ in\ this\ event's\ state}{\#\ of\ total\ subscribed\ events}$$

$$F_4 = \frac{\#\ of\ similar\ users\ subscribed\ to\ this\ event}{\#\ of\ total\ similar\ users} \sum_{j=0}^{3} F_j$$

All events are then ordered by *F* in descending order and the *n* events with the greatest *F* are displayed to the user, where *n* is an integer input to the algorithm. As a result of this algorithm, the range of *F* is [0, 100]. The pseudocode of this algorithm's implementation looks like the following:

```
foreach(event in allEvents.Reverse()) {
      Calculate F0
      Calculate F1
      Calculate F2
      Calculate F3
      Calculate F4
      F = F0 + F1 + F2 + F3 + F4
}
allEvents.OrderByDescending(event => event.Fit)
return first n events in allEvents
```

This recommendation algorithm is good at generating events that are similar to the ones that the user has already subscribed to. Additionally, due to the collaborative nature of F_4 as well as the way that it scales with the other factors, this algorithm is also capable of recommending events that are quite different from what the user has already subscribed to, but even these events have a decent chance of piquing the user's interest. This is effective because variability in the kinds of events that are recommended to the user adds interest to our application's recommended section.

For Each Event

This recommendation algorithm takes an integer n and an event as input and then generates n recommendations based on the event's data. This algorithm is based on item-to-item collaborative filtering, but determines the similarity between the input event and all

other events online instead of offline. Similar to our 'For Each User' recommendation algorithm, this algorithm generates the recommendations by generating a fitness score for every event and then ranks the events based on this fitness score. The highest-ranking events are the ones displayed to the user. The fitness score for this algorithm is intended to represent how much a user that is interested in the input event would be interested in another event and is generated based on the following four weighted factors:

i	Factor	Weight (fitness score)
0	The percentage of keywords in this event's title that are also in the input event's title	15 * this percentage
1	Whether the event and the input event are of the same category	20
2	Whether this event and the input event are in the same state	15
3	The percentage of same users that have subscribed to this event	30 * this percentage

Table 2: Fitness Score Factors for 'For Each Event' Algorithm

Same users are defined as users who have subscribed to both this event and the input event. The fitness score *F* for each event other than the input event, where F_i is the score contributed by factor i, is calculated as:

$$F = \sum_{i=0}^{3} F_i$$

$$F_0 = 15 * \frac{\#\ of\ matching\ keywords\ in\ this\ event's\ title}{\#\ of\ keywords\ in\ this\ event's\ title}$$

$$F_1 = 20 * (1\ if\ this\ event\ is\ in\ the\ same\ category,\ 0\ if\ not)$$

$$F_2 = 15 * (1\ if\ this\ event\ is\ in\ the\ same\ state,\ 0\ if\ not)$$

$$F_3 = 30 * \frac{\#\ of\ same\ users\ subscribed}{\#\ of\ users\ subscribed\ to\ input\ event}$$

All events are then ordered by *F* in descending order and the *n* events with the greatest *F* are displayed to the user, where *n* is an integer input to the algorithm. As a result of this algorithm, the range of *F* is [0, 80]. The pseudocode of this algorithm's implementation looks like the following:

```
foreach(event in allEvents.Reverse()) {
      Calculate F0
      Calculate F1
      Calculate F2
      Calculate F3
      F = F0 + F1 + F2 + F3
}
allEvents.OrderByDescending(event => event.Fit)
return first n events in allEvents
```

This recommendation algorithm is good at generating events that are similar to the input event. Also, due to the collaborative nature of F_3, this algorithm is capable of recommending events that are quite different from the input event. However, even these events are likely to be of interest to a user that is interested in the input event. Much like the variability of the events recommended by the 'For Each User' algorithm, the variability of the events recommended by this algorithm increases its effectiveness.

Design Decisions

Events

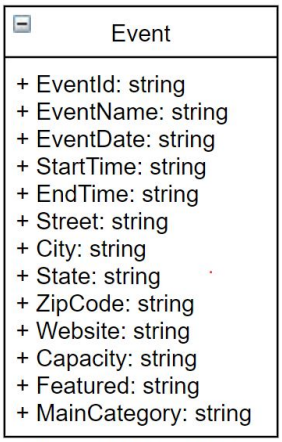

Figure 1.1: Event class

Since our project revolves around the recommendation of events, it was very important that a lot of thought went into the designing of what an event is. In order to do this, we first decided what the structure of the events would look like based on what data would be used to make recommendations and what information would be useful for a potential user. As a result, we ended up creating a simple and generalized event data structure as seen in figure 1.1. This model has been used for all created events and has worked well for our purposes. However, there is definitely a lot of room for improvement. For example, additional event classes could be created and extend the current event class with more data. This would allow us to be more specific about events and include additional details based on the type of events such as genres for movies, music, and video games which could also be used in the recommendation algorithm to make even better recommendations.

Another decision we had to make was the source from where the event data comes from. After a lot of experimentation, we decided to create a service that would randomly generate a lot of events and another service to specify the details of an event. We primarily made this decision because we wanted to be able to have some level of control and predictability in the events being used for the algorithm in order for us to test the overall system. This has worked very well for us so far because we were able to generate the data based on payloads we put together using data from existing top charts to create realistic events. Alternatively, we could have utilized APIs from places such as Seat Geek, Ticketmaster, and

Rotten Tomatoes, but we had difficulty finding any other sources with good data and were concerned about the variety that was offered.

Users

Similar to events, we needed to build users for the web application. While we were working on the structure for a user, we kept it as simple as possible. This resulted in the structure shown in figure 1.2. In addition, we also created the functionality to randomly generate a lot of different users using a large set of data. Overall, this worked out quite well for us but to have real users in the future we will need to include more information in this model as well as user authentication and authorization functionality.

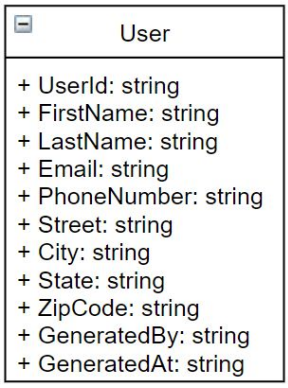

Figure 1.2: User class

Architecture

The backend API acts as a transformative intermediary between the frontend UI and the database. It queries information from the database tables, performs logic on the information, and transforms it into more useful and consumable information in JSON format. It is built in C# as a .Net Core application that utilizes MVC and n-tier architecture to account for separation of concerns by splitting the project into contracts, presentation, logic, and infrastructure layers as shown in figure 1.3. However, the 'View' part of the MVC architecture was separated into its own frontend project. The RESTful

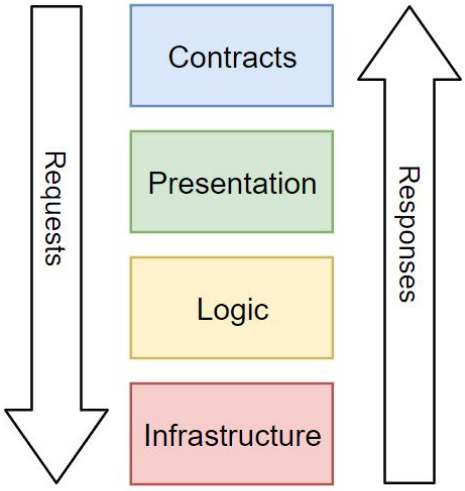

Figure 1.3: N-Tier Architecture

API controllers are split up based on their core functionality such as categories, events, random, and users. This has worked out well for us due to the collaborative nature of the project.

We built our frontend UI using ReactJS because it allows for the creation of extremely dynamic web pages and the componentization of web page elements, making pieces of our code very reusable. To make requests to our backend API, we utilized Axios because it is a very flexible HTTP client that works well with ReactJS. The structure of the application as a whole and how the different pieces interact with each other is shown in figure 1.4.

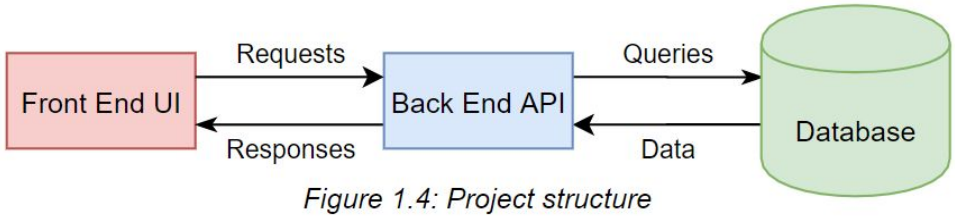

Figure 1.4: Project structure

Backend API

Categories

The categories-related endpoints of our API are used to interact with the Categories table of our database. This is the table where the categories that events can belong to are stored.

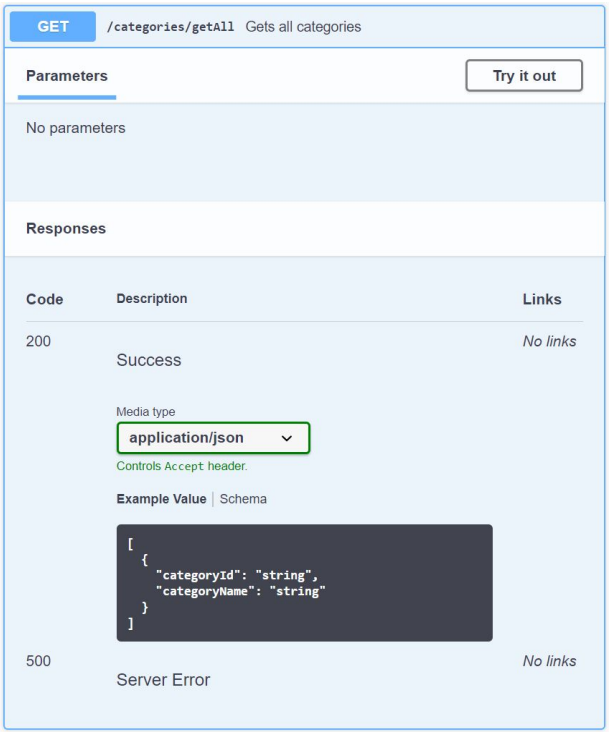

Figure 1: Get All Categories Endpoint

The ‘Get All Categories’ endpoint (*Figure 1*) returns all of the categories in the database and is used frequently within the UI for populating menus that use categories.

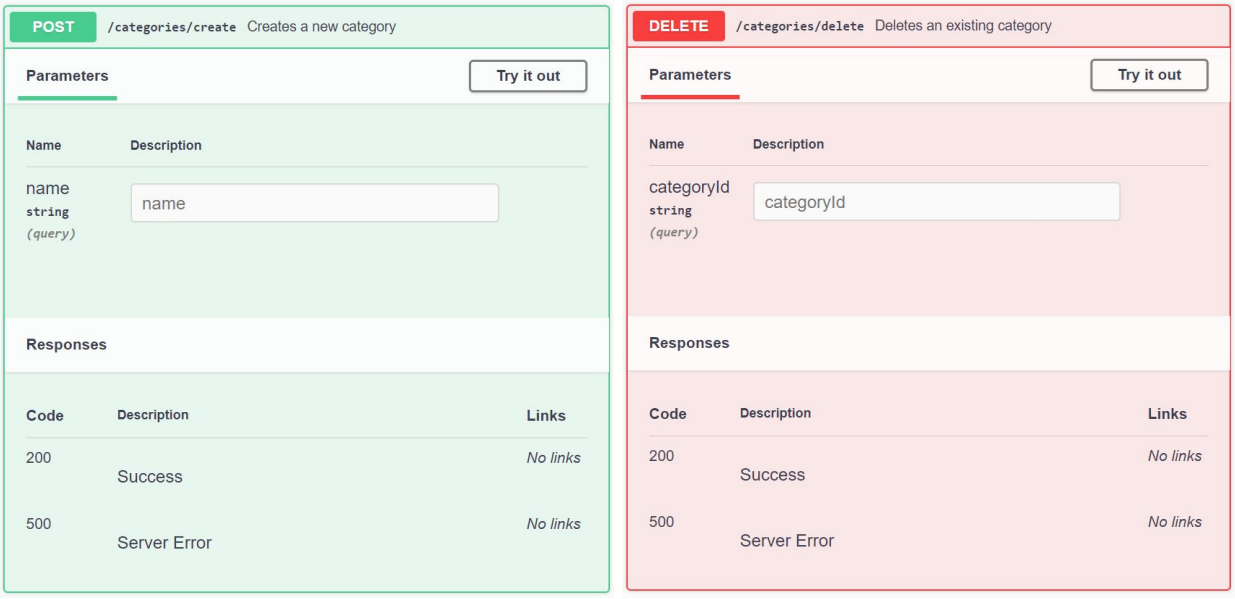

Figure 2: Create (left) and Delete (right) a Category Endpoints

The ‘Create a Category’ and ‘Delete a Category’ endpoints (*Figure 2*) add a new category to the database or delete an existing category from the database.

Events

The event-related endpoints in our API are used for interacting with events, which involves the Events table, Users table, and UserEvents (subscriptions) table in our database. The Events table stores all of the events, the Users table stores all of the users, and the UserEvents table stores all of the subscription relationships between users and events. Many of the event-related endpoints also utilize our recommendation algorithms to generate their responses.

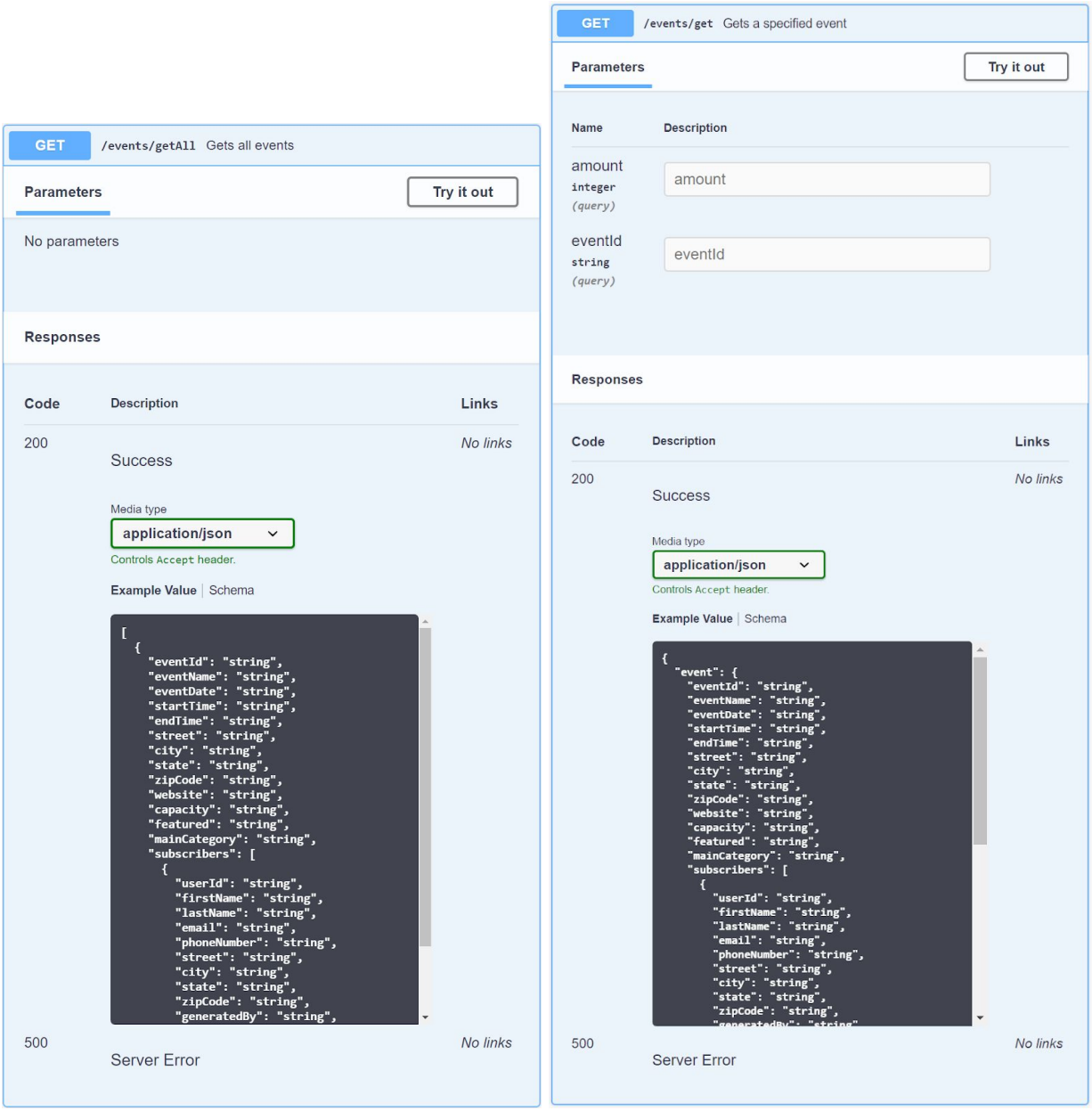

Figure 3: Get All Events Endpoint (left) and Get an Event Endpoint (right)

The ‘Get All Events’ endpoint (*Figure 3*) retrieves all of the events in the Events table. The ‘Get an Event’ endpoint (*Figure 3*) gets a specific event from the Events table, based on its eventId. These are used on the browse page, events page, and manage events page to get the latest information from the system.

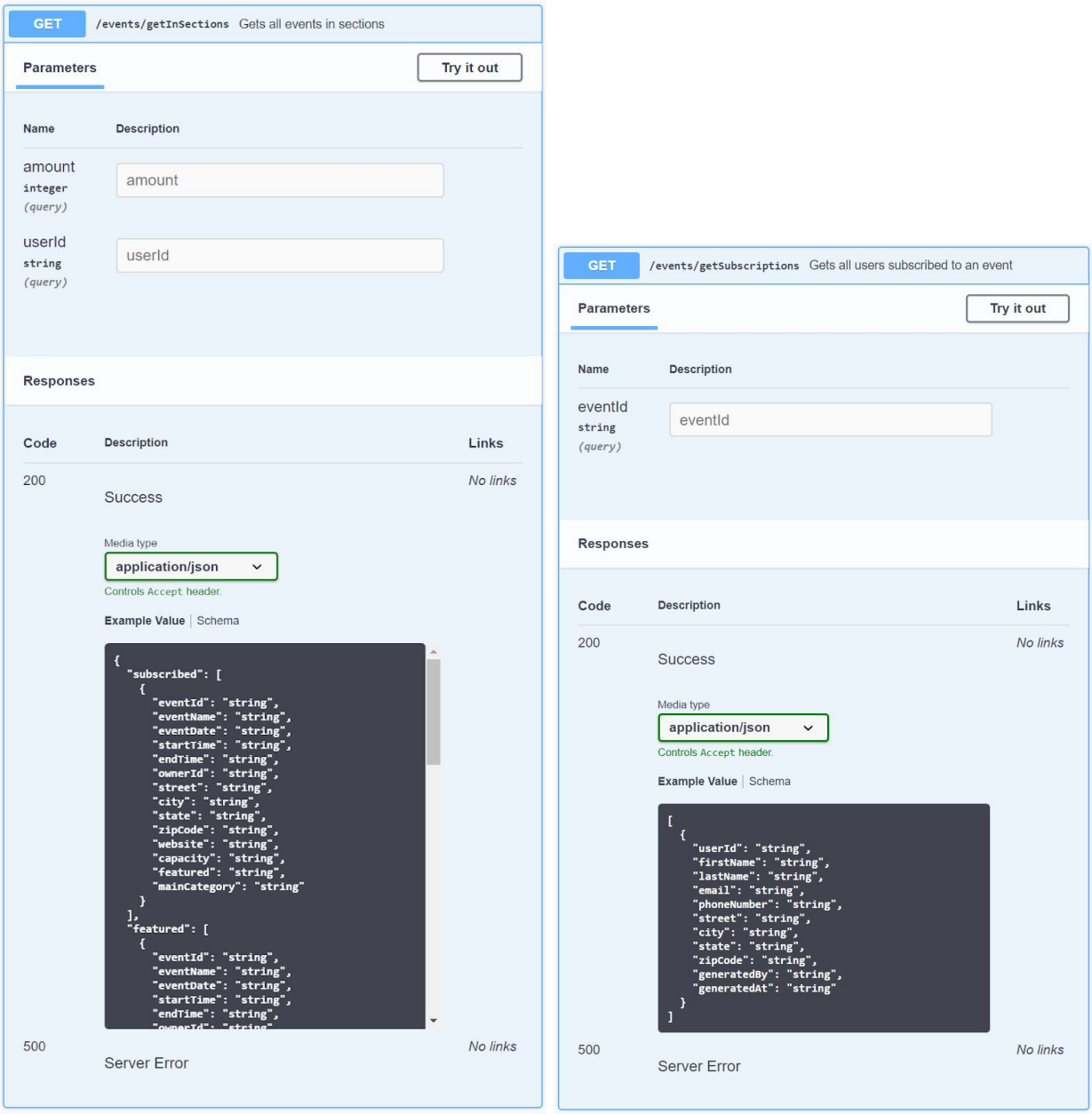

Figure 4: Get Events in Sections Endpoint (left) and Get Event Subscribers Endpoint (right)

The ‘Get Events in Sections’ endpoint (*Figure 4*) retrieves and responds with all of the events that an input user has subscribed to, all featured events, all events recommended for the input user (based on the ‘For Each User’ algorithm), and the most subscribed-to events to display on the home page. The ‘Get Event Subscribers’ endpoint (*Figure 4*) responds with all of the users that have subscribed to the input event.

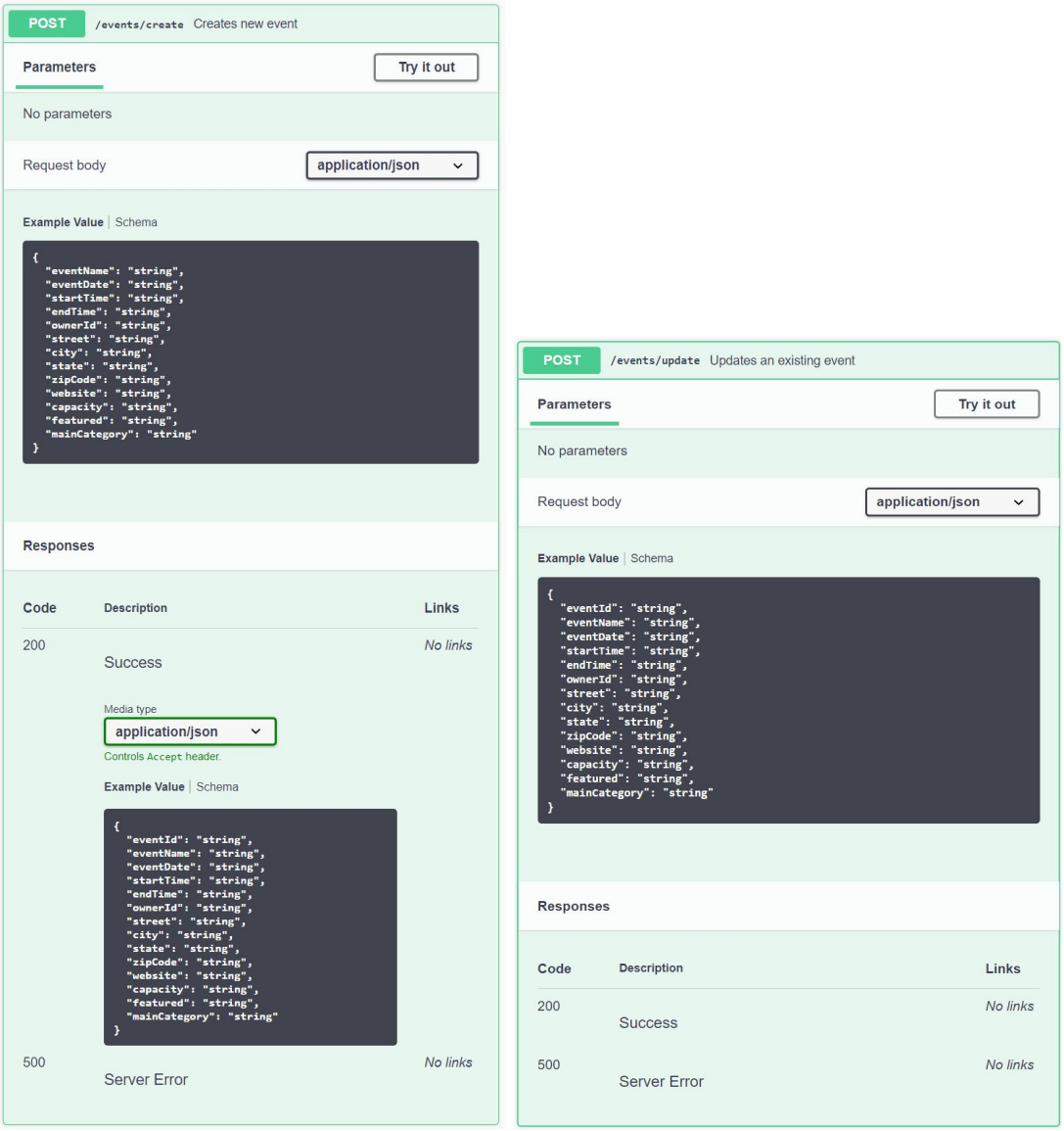

Figure 5: Create (left) and Update (right) an Event Endpoints

The ‘Create an Event’ and ‘Update an Event’ endpoints (*Figure 5*) allow the user to create a new event in the Events table or update an existing event in the Events table. As of right now, these are primarily used in our admin page to manage our events in the database.

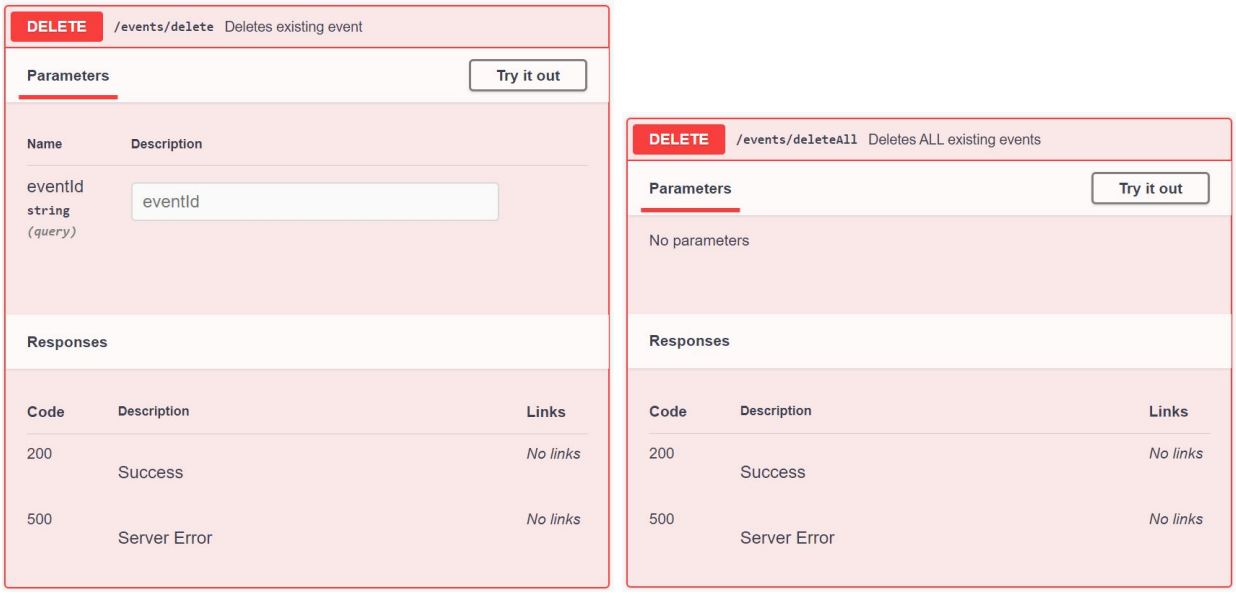

Figure 6: Delete an Event (left) and Delete All Events (right) Endpoints

The 'Delete an Event' and 'Delete All Events' endpoints (*Figure 6*) allow the user to delete a single event from the Events table or delete all events from the Events table.

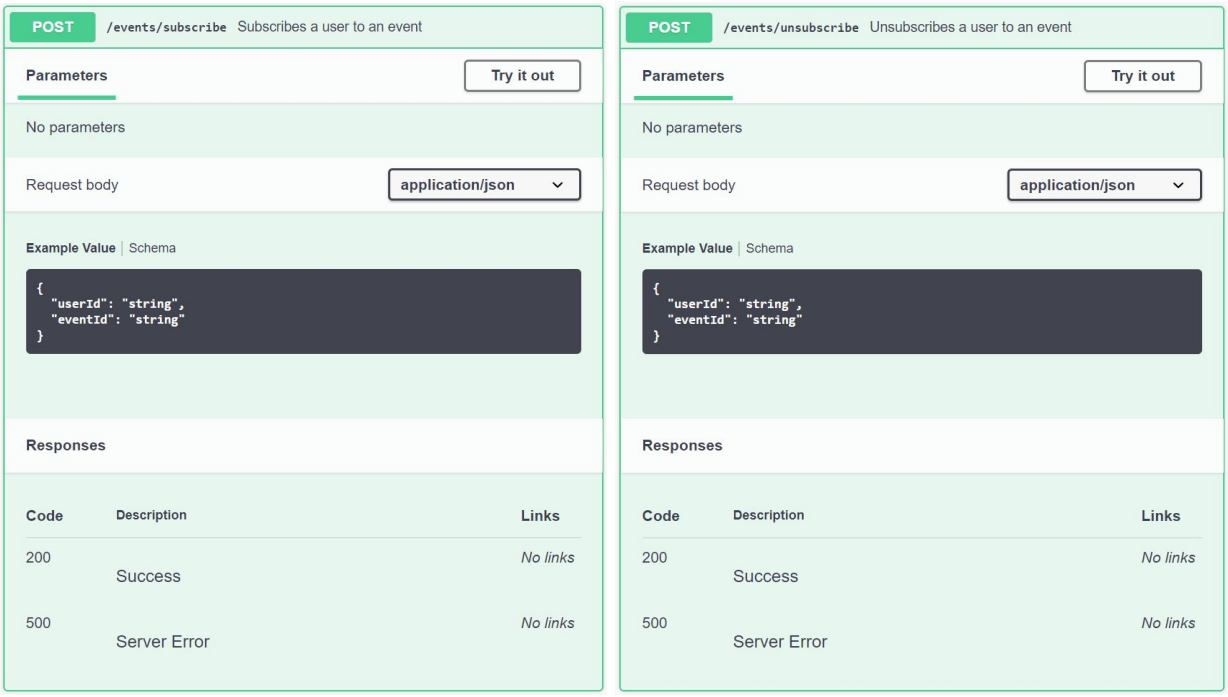

Figure 7: Subscribe (left) and Unsubscribe (right) Endpoints

The ‘Subscribe’ and ‘Unsubscribe’ endpoints (*Figure 7*) either add or remove a subscription from the UserEvents (subscriptions) table. This is used in the event pages for users who are logged in to either subscribe or unsubscribe to an event.

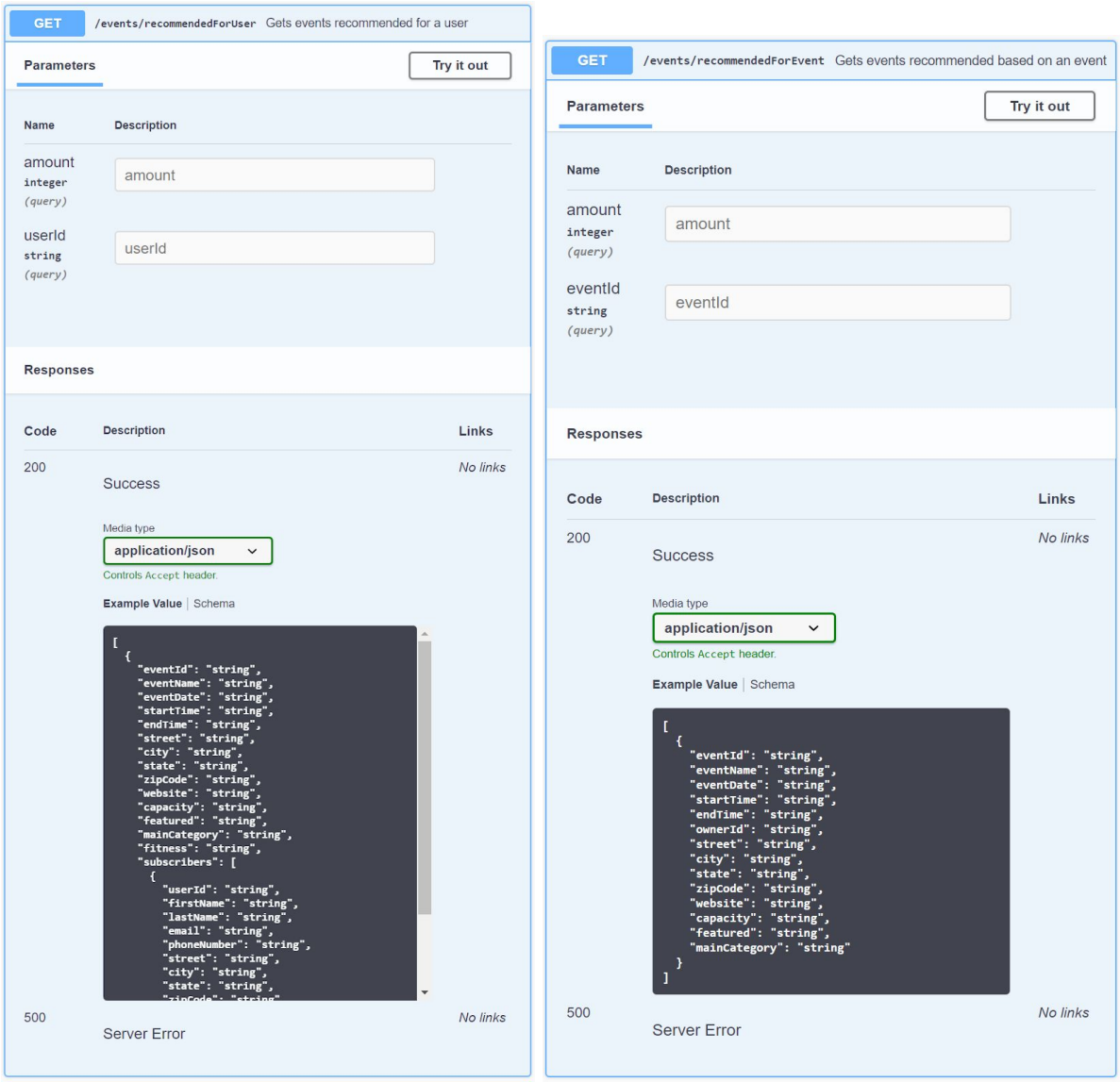

Figure 8: ‘For Each User’ (left) and ‘For Each Event’ (right) Recommendation Endpoints

The ‘For Each User’ recommendation endpoint (*Figure 8*) takes an integer and a user as input and responds with the resulting events from providing these input values to the ‘For Each User’ recommendation algorithm. The ‘For Each Event’ recommendation endpoint (*Figure 8*)

takes an integer and an event as input and responds with the resulting events from providing these input values to the ‘For Each Event’ recommendation algorithm.

Random

The ‘Create Random Users’ and ‘Create Random Events’ endpoints as shown in *Figure 9* are used to generate random realistic data for testing our application. Both utilize a series of large payloads in order to create a variety of test data for us to use.

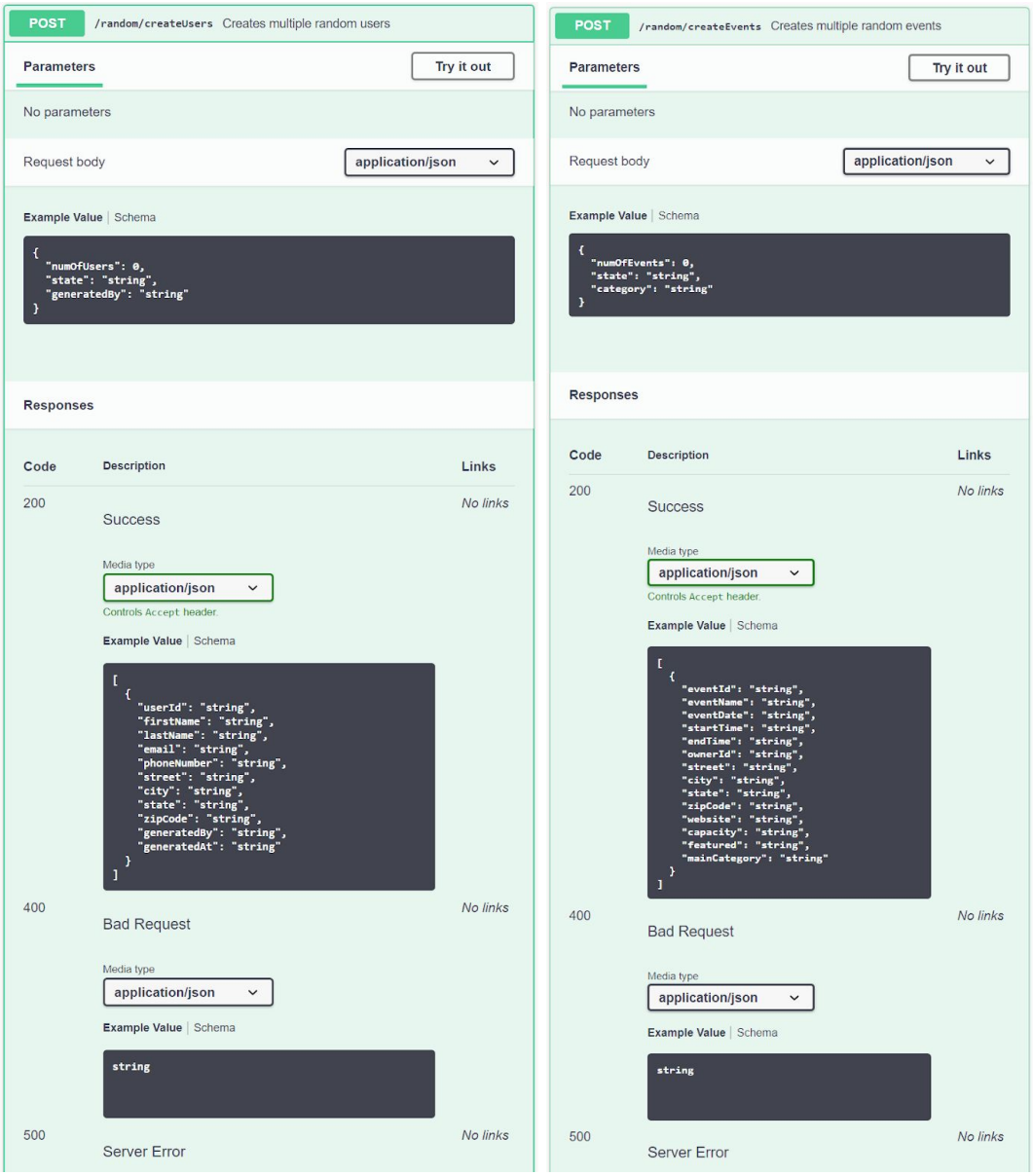

Figure 9: Create Random Users (left) and Random Events (right) Endpoints

Users

The users-related endpoints interact with the Users table and the UserEvents (subscriptions) tables of our database.

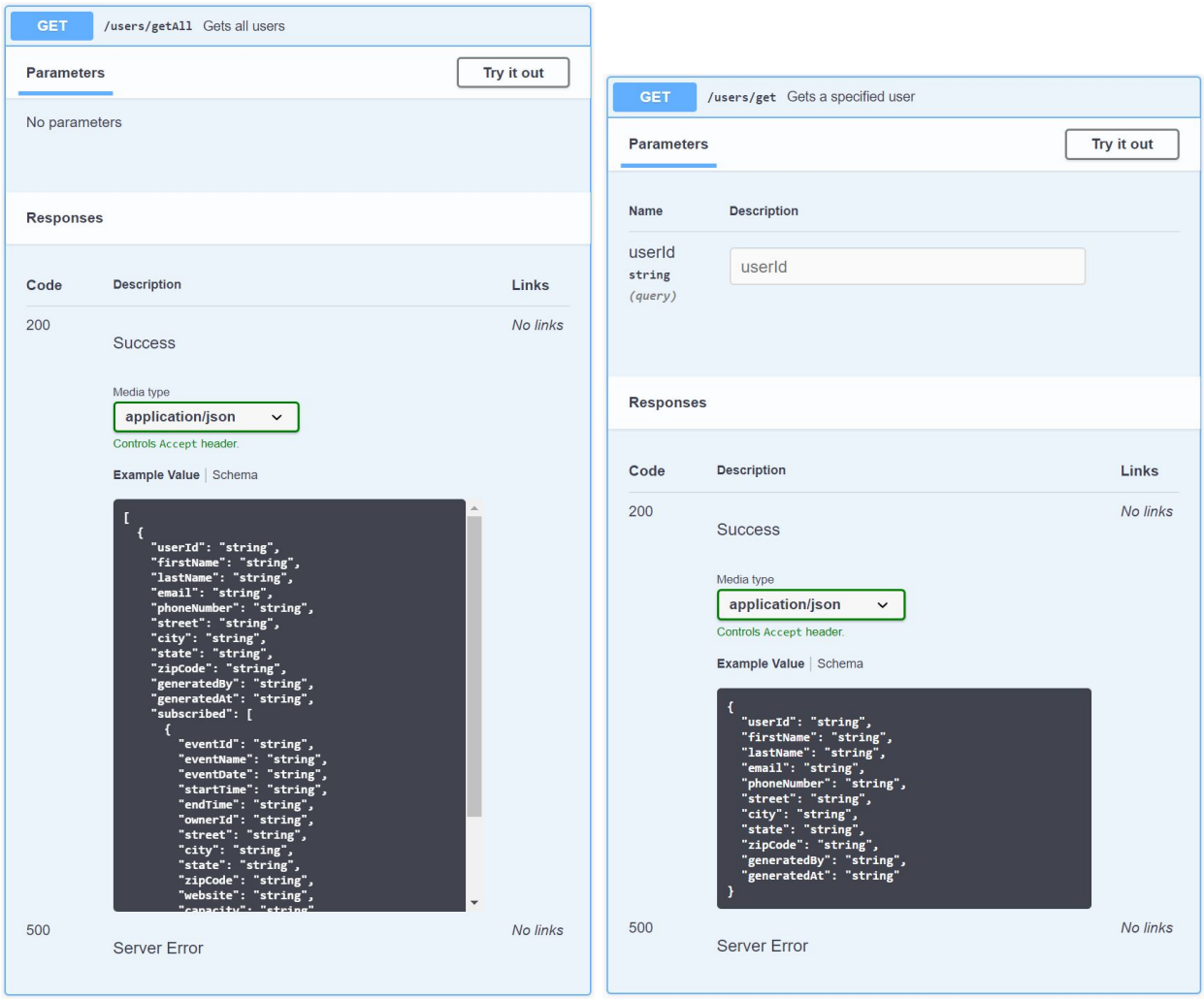

Figure 10: Get All Users (left) and Get a User (right) Endpoints

The ‘Get All Users’ and ‘Get a User’ endpoints (*Figure 10*) retrieve all of the users in the Users table or a single user in the Users table, respectively. This is used to select a user to login as in the UI in order to simulate the use of the website from their point of view.

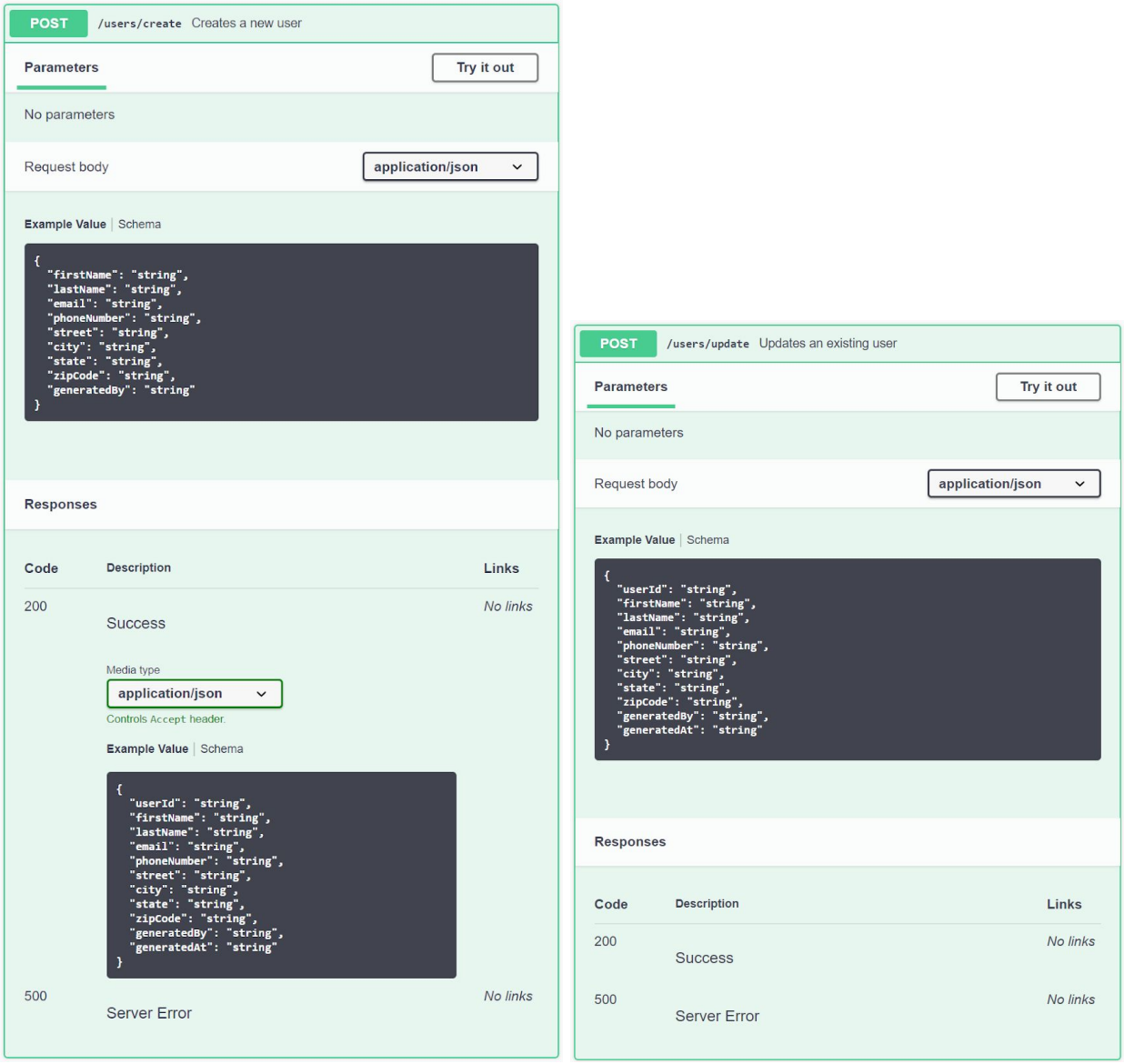

Figure 11: Create (left) and Update (right) Users Endpoints

The ‘Create User’ and ‘Update User’ endpoints (*Figure 11*) either create a new user in the Users table or update an existing user in the Users table. These are primarily used in our admin page to manage our users in the database.

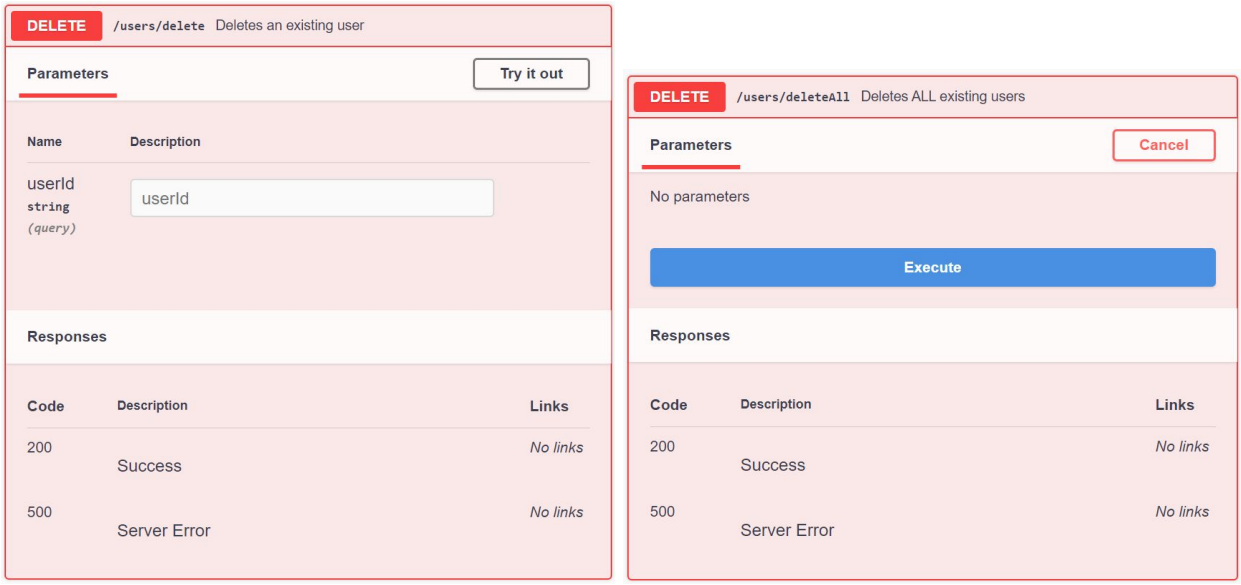

Figure 12: Delete a User (left) and Delete All Users (right) Endpoints

The ‘Delete a User’ and ‘Delete All Users’ endpoints (*Figure 12*) either delete a specific user from the Users table or delete all existing users from the Users table, respectively.

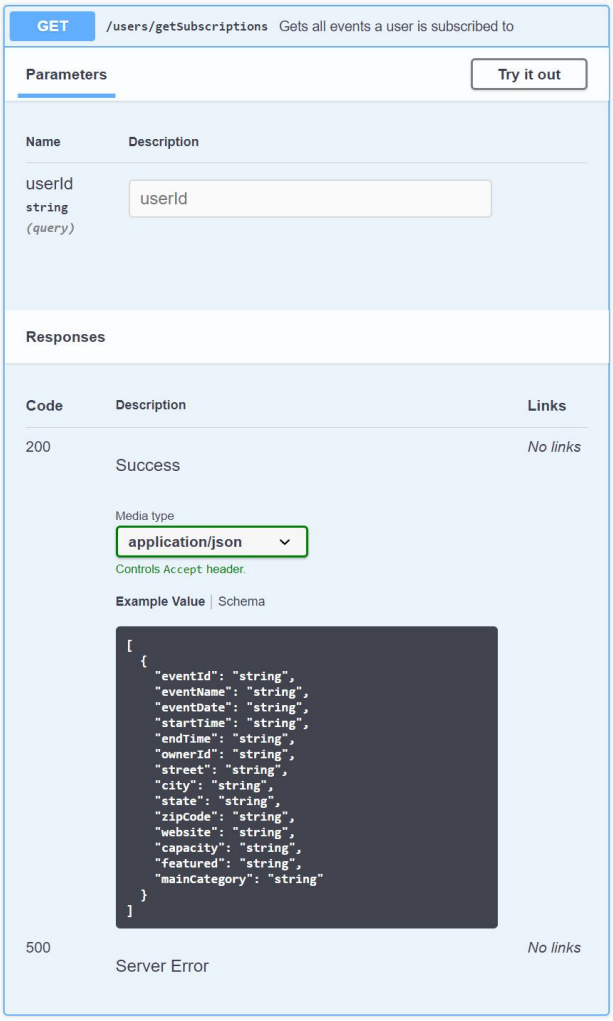

Figure 13: Get a User's Subscriptions Endpoint

The 'Get a User's Subscriptions' endpoint (*Figure 13*) retrieves and responds with all events that the input user has subscribed to.

Frontend UI

When the frontend UI is first launched, the homepage will look like *Figure 14* since no user is logged-in. On this page, the user can see the events that have been marked as 'featured' and the events with the most subscribers. If the user clicks the 'Login' button at the

top right of the screen, the login modal will pop up as shown in *Figure 15*. This modal displays every user and gives the option to login with any one of them. After the 'Login' button for one of the users is clicked, the login modal will close and the homepage will look like *Figure 16*, with the 'My Events' and 'Recommended Events' sections now being included in the view. The 'Recommended Events' section in this view utilizes the 'For Each User' recommendation algorithm, with the logged-in user as input.

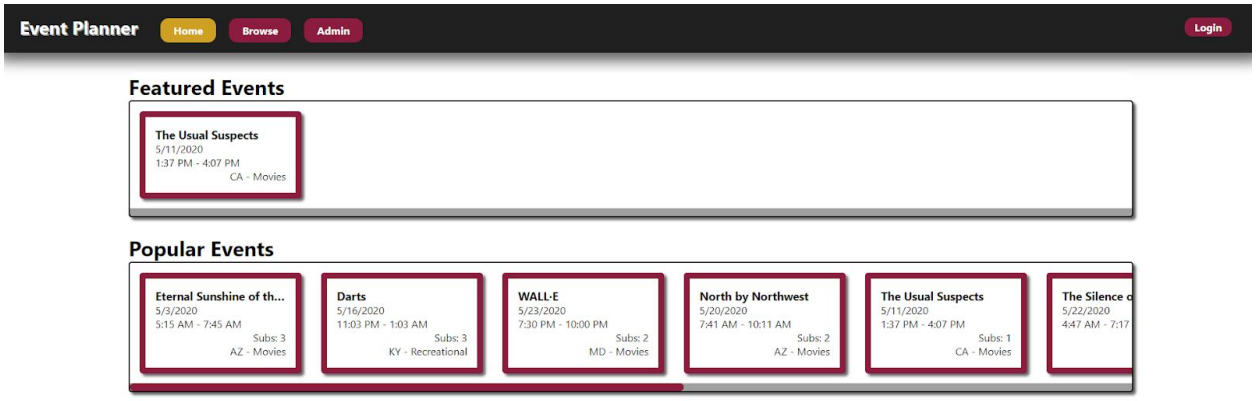

Figure 14: Logged-Out Homepage

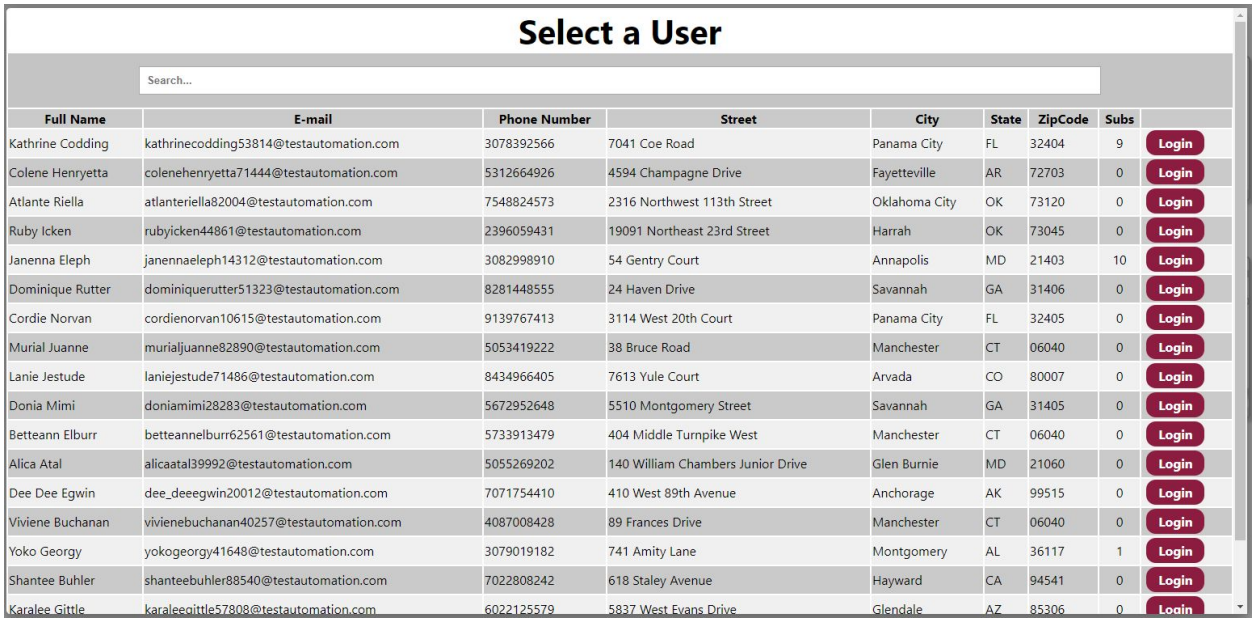

Figure 15: Login Modal

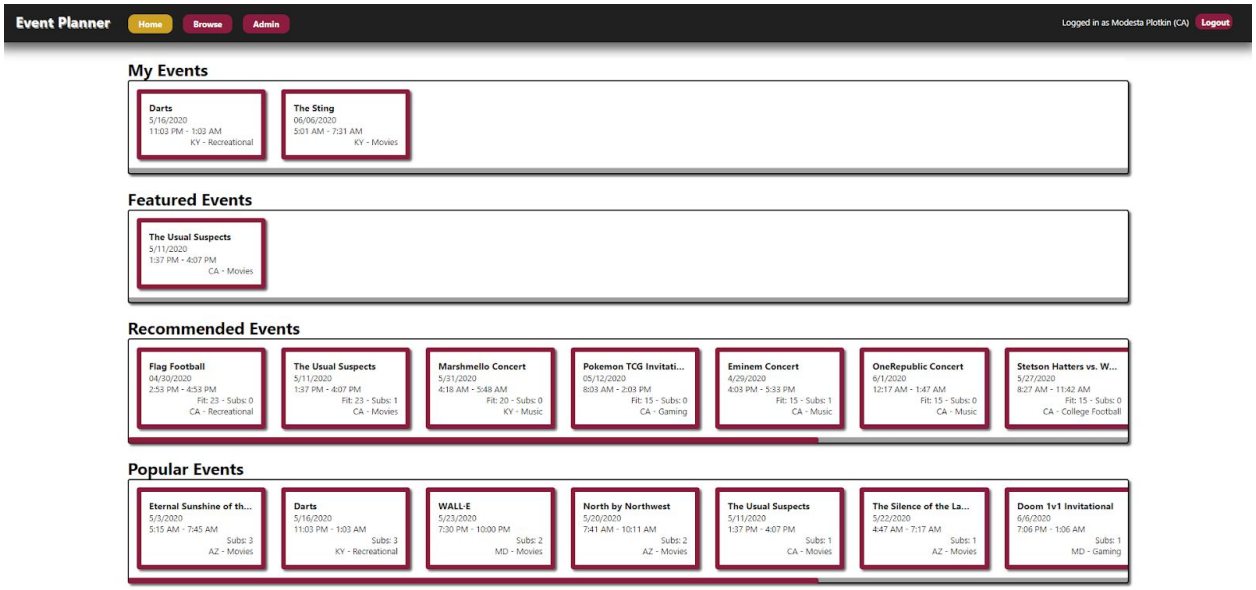

Figure 16: Logged-In Homepage

If the user clicks on one of the event cards in any of the rows on the homepage, they will be redirected to that event's event details page, as shown in *Figure 17* if they are logged-out or

Figure 18 if they are logged-in. The difference between these two views of this page is that the user can subscribe to or unsubscribe from the event if they are logged in, but not if they are logged out. In both views of the event details page, the user can see the event's name, time, location, category, and other events that are similar to this event. The 'Similar Events' row uses the 'For Each Event' recommendation algorithm to generate the recommended events, with the page's event as the input.

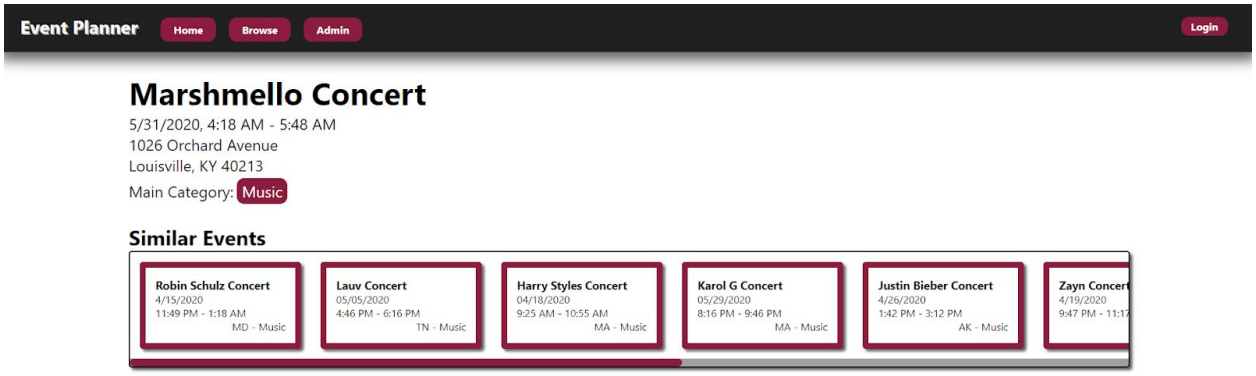

Figure 17: Logged-Out Event Details Page

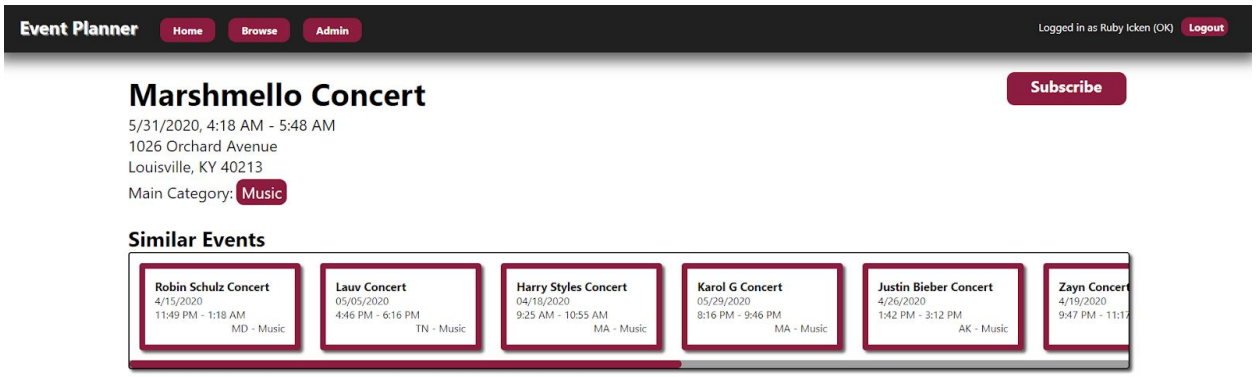

Figure 18: Logged-In Event Details Page

If the user clicks on the ‘Admin’ button in the header, they will be redirected to the ‘create specific users’ page as shown in *Figure 19*, where the user can input the attributes of a user to add a new user to the database. If the user hovers over the ‘Users’ button and clicks the ‘Create Random’ button in the dropdown that appears, they will be redirected to the ‘Create Random Users’ page as shown in *Figure 20*. This process allows us to make multiple users with automatically generated data. Similarly, if the user hovers over the ‘Users’ button and clicks the ‘Manage’ button in the dropdown that appears, they will be redirected to the ‘Manage Users’ page as shown in *Figure 21*. This page allows the user to search through all of the users in the database. Clicking the ‘Delete’ button will delete that user from the database. As shown in *Figure 22*, clicking the ‘View’ button displays the information of that user, and clicking the ‘Edit’ button allows you to change the attributes of that user in the database.

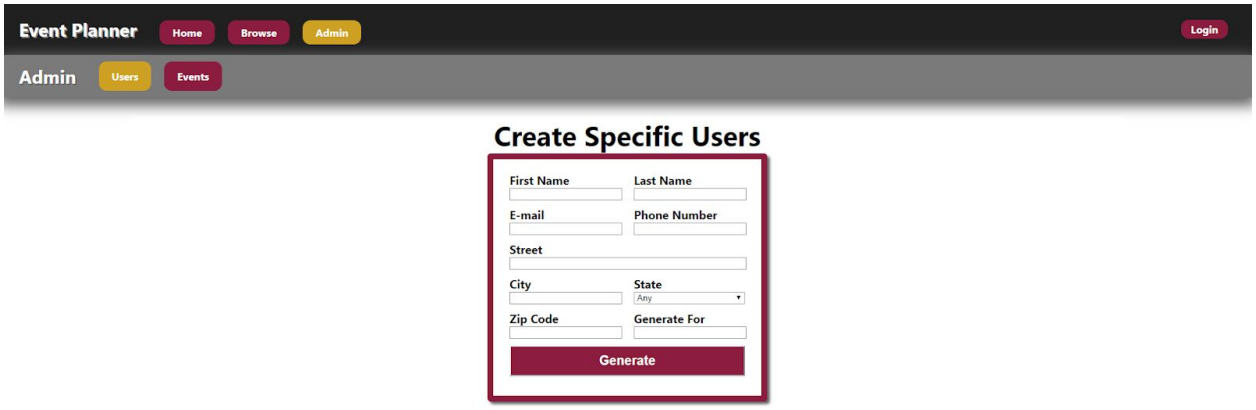

Figure 19: Create Specific Users Page

Figure 20: Create Random Users Page

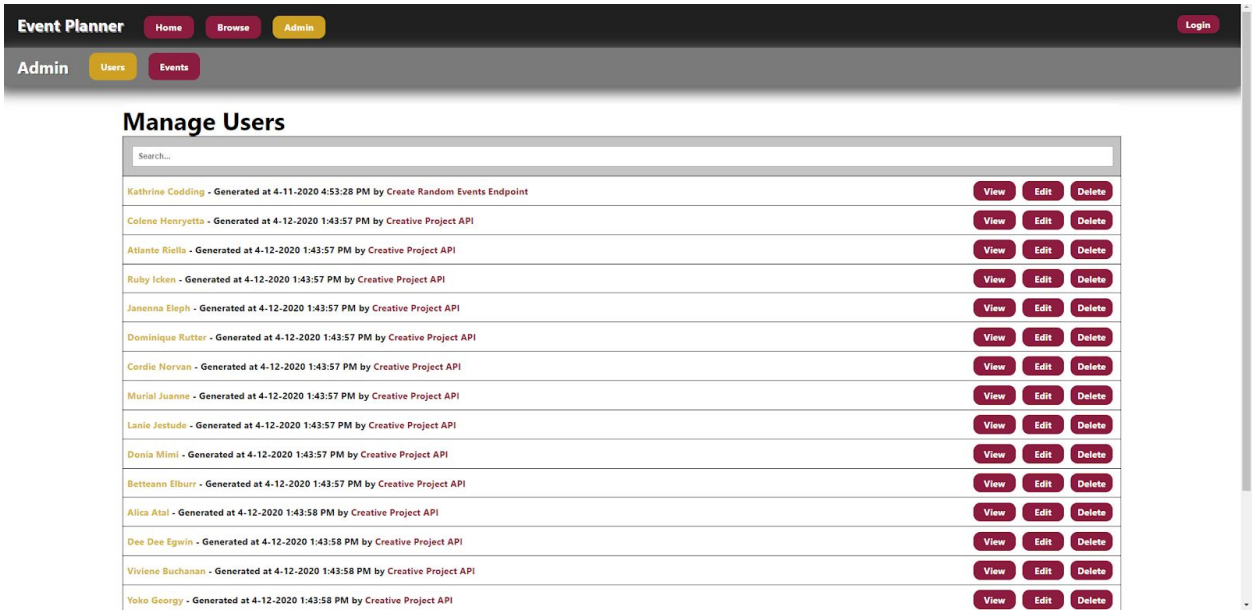

Figure 21: Manage Users Page With Nothing Selected

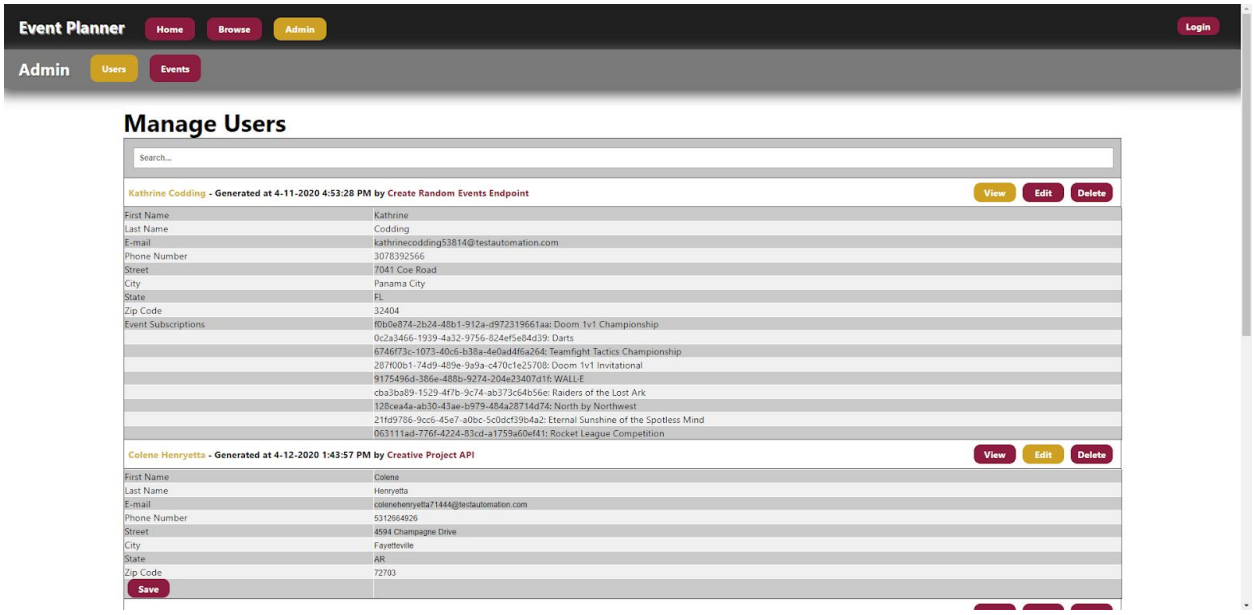

Figure 22: Manage Users Page With Users Selected

If the user hovers over the ‘Events’ button in the admin page and then clicks on the ‘Create Specific’ button in the dropdown that appears, they will be redirected to the ‘Create Specific Events’ page as shown in *Figure 23*. This page allows the user to input the attributes of an event to add a new event to the database. If the user hovers over the ‘Events’ button and clicks the ‘Create Random’ button in the dropdown that appears, they will be redirected to the ‘Create Random Events’ page as shown in *Figure 24*. This process allows us to make multiple events with automatically generated data. Similarly, if the user hovers over the ‘Events’ button and then clicks on the ‘Manage’ button in the dropdown list that appears, they will be redirected to the ‘Manage Events’ page as shown in *Figure 25*, where the user can search through all of the events in the database. If the user clicks the ‘Delete’ button, that event will be deleted from the database. As shown in *Figure 26*, clicking the ‘View’ button displays the information of that event, and clicking the ‘Edit’ button allows the user to change the attributes of that event in the database.

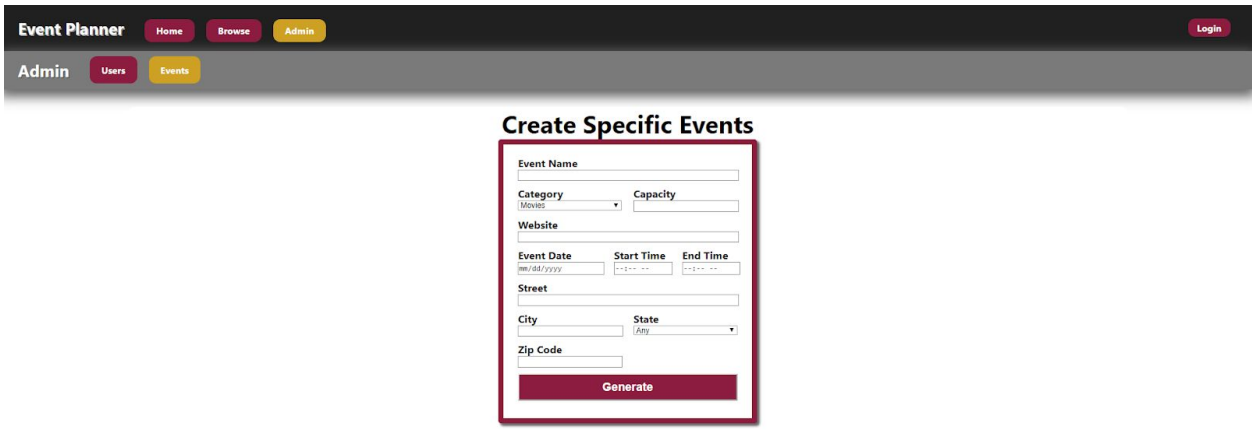

Figure 23: Create Specific Events Page

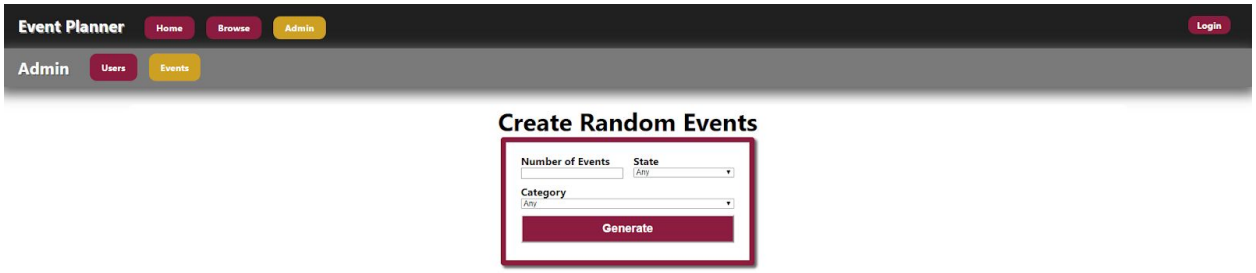

Figure 24: Create Random Events Page

Figure 25: Manage Events Page With Nothing Selected

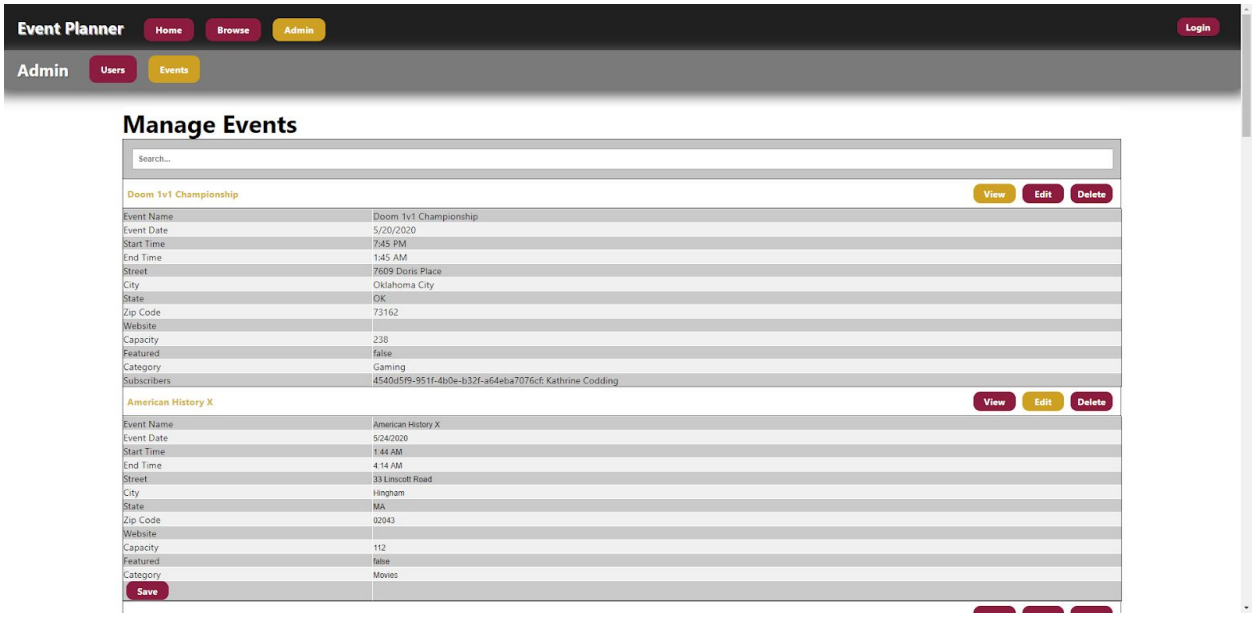

Figure 26: Manage Events Page With Events Selected

If the user clicks on the ‘Browse’ button in the header, they will be redirected to the ‘Browse Events’ page as shown in *Figure 27*, where the user can search through all the events

and filter through the results using the category and the state as options. In addition, this page also highlights events that a user is already subscribed to if they are logged in as shown in *Figure 28*.

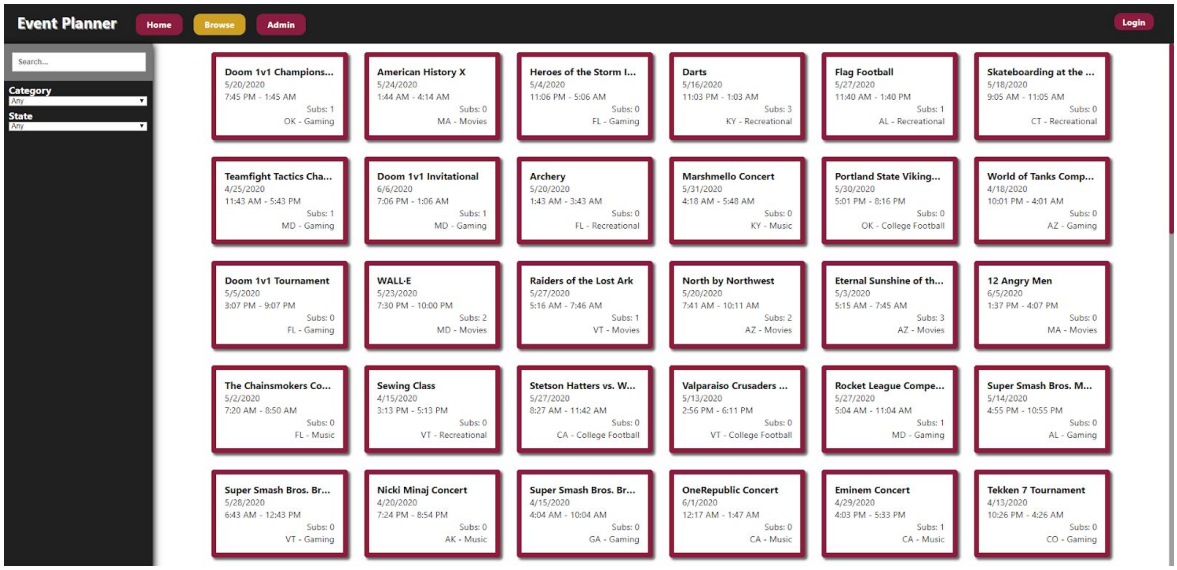

Figure 27: Browse Events Page Logged Out

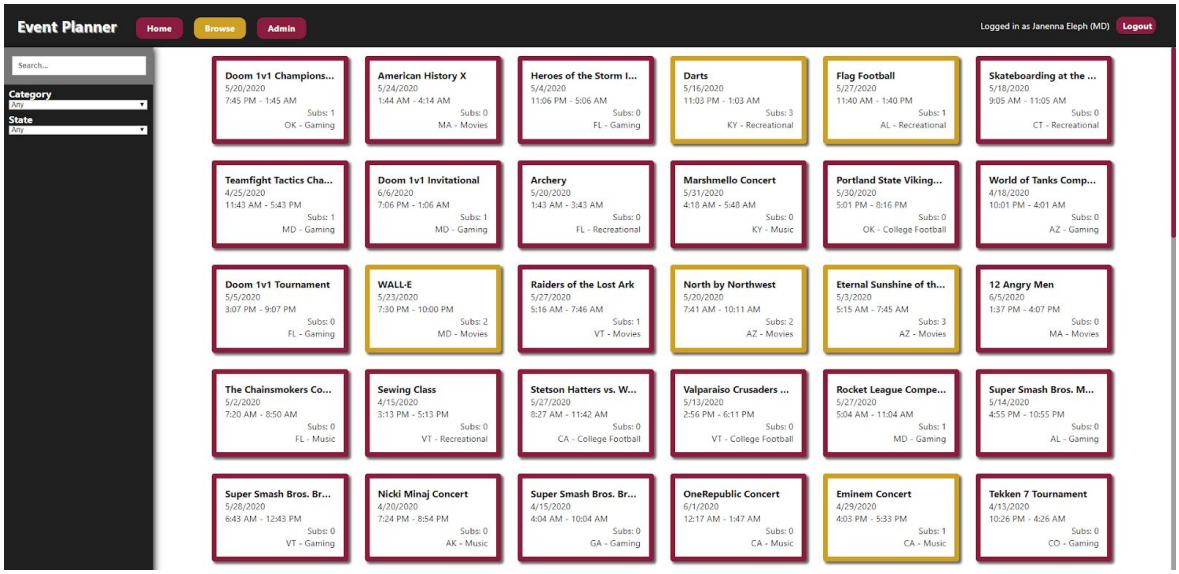

Figure 28: Browse Events Page Logged In

Future Work

There are many features that could be added to enhance this project, and in this section we will discuss some of the features that we believe would enhance this project the most. One of these features is a friends system, where users can add each other as friends and see which of their friends is attending each event. If this feature were implemented, it could also be added as a factor for the recommendation algorithms' fitness scores. This factor would increase the variability of the events that the algorithms recommend, which would make the algorithms more effective at recommending events that are likely to be of interest to the user. Additionally, allowing users to see which of their friends are planning to attend an event may influence those users to also attend the event.

Another feature that would greatly enhance this project is a scheduling system. On account creation, users could input their schedule and the recommendation algorithms would only recommend events that fall within this schedule. This would act as a great pre-filtering strategy, which would improve our algorithms' performance and scalability. This feature combined with the friends system would also enable users to collaborate on the scheduling of group events.

In addition, adding the ability for real users to create accounts with an authentication system and Google Calendar integration would be a large step towards making our project publicly usable. The natural next step would be to allow these users to create real events and also automatically pull real events from external sources in place of our emulated data.

There are many ways in which we could improve our recommendation algorithm. The most valuable step to take towards this would be to make this application publicly usable and collect feedback from our users. This feedback could either be in the form of voluntary feedback

given directly from our users, or it could be in the form of analytical logging. Additionally, it would be very useful to research the effectiveness of several alterations of our algorithm via A/B testing. Alterations as simple as changing the weights of each of the factors in our algorithms could potentially make a great impact, but alterations as complex as implementing some form of machine learning could also provide value. A candidate-generation neural network or pre-filtering strategies could improve the performance and scalability of our algorithms, which are most likely their biggest weaknesses.

Conclusion

Overall, the Group Event Planner App has the potential to make the creation, sharing, and discovering of events very fast and easy. To facilitate the testing of the recommendation algorithms, the web app currently works as a representation of the final product using randomized test data for simulating and demonstrating the web app from the perspective of a user. With a little more time, it could be transitioned into a fully usable application that would be able to provide real value to real people. The project in its current state functions well as a foundation for our overall goals, as it allows for the creation and management of both events and users while also utilizing our proposed recommendation algorithms to suggest events to our users.

References

Chen, Yinong. *Service-Oriented Computing and System Integration*. 6th ed., Kendall/Hunt, 2018.

Cornelis, Chris, et al. “A Fuzzy Relational Approach to Event Recommendation.” *Proceedings of the 2nd Indian International Conference on Artificial Intelligence, IICAI*, 2005, pp. 2231–2242.

Covington, Paul, et al. “Deep Neural Networks for YouTube Recommendations.” *Proceedings of the 10th ACM Conference on Recommender Systems - RecSys 16*, 2016, doi:10.1145/2959100.2959190.

Gomez-Uribe, Carlos A., and Neil Hunt. “The Netflix Recommender System.” *ACM Transactions on Management Information Systems*, vol. 6, no. 4, 2016, pp. 1–19., doi:10.1145/2843948.

IR. “SeatGeek – Creating Price Transparency in the Ticket Market.” *Digital Innovation and Transformation*, digital.hbs.edu/platform-digit/submission/seatgeek-creating-price-transparency-in-the-ticket-market/.

Linden, G., et al. “Amazon.com Recommendations: Item-to-Item Collaborative Filtering.” *IEEE Internet Computing*, vol. 7, no. 1, 2003, pp. 76–80., doi:10.1109/mic.2003.1167344.

Murphy, Shawn M., et al. “Event recommendation service.” US20100325205A1, United States Patent and Trademark Office, 17 June 2009.

Qin, Zhen, et al. “A Scalable Approach for Periodical Personalized Recommendations.” *Proceedings of the 10th ACM Conference on Recommender Systems - RecSys 16*, 2016, doi:10.1145/2959100.2959139.

“YouTube for Press.” *YouTube*, YouTube, www.youtube.com/about/press/.

Zambrano, Brian Richard, and Luke O'Daniel Groesbeck. “Social event recommendations.” US8700540B1, United States Patent and Trademark Office, 29 November 2010.