TigerNest: Event Organizer and Host Flow
Simplifying Overnight Host-Visitor Pairing at Princeton

Niranjan Shankar
Princeton University, Class of 2020
Spring 2019 Independent Work Project
Advisor: Dr. Robert Dondero

Abstract

Currently, organizing overnight events at Princeton is a cumbersome process for student organization leaders (event organizers), as they are burdened with manually pairing Princeton student volunteers (hosts) with overnight guests (visitors). This paper will discuss a web application developed over the course of the semester, called TigerNest, to address this inconvenience and simplify the process of facilitating events and finding overnight accommodation at Princeton. Through TigerNest, event organizers will be able to create events that hosts and visitors will be able to sign up for. In this sign up process, hosts volunteer their rooms and specify their room capacity and gender preferences, and visitors can sign up for their rooms as per their gender preferences. This approach was chosen in order to meet the various requirements of hosts and visitors while also giving visitors the flexibility to choose their accommodation overnight. This web application consists of three separate paths or "flows": the event organizer, host, and visitor flows. This paper will discuss the event organizer and host flows, which was the primary focus of this project, though some aspects of the visitor flow will be discussed for clarification. This project was done in conjunction with another independent project from this semester - TigerNest: Visitor Flow.

1. Introduction

Overnight accommodation is an integral, yet extremely complicated, aspect of organizing successful, large-scale events at Princeton. In addition to facilitating the event itself, many student organizations leaders are also burdened with matching overnight visitors with Princeton hosts manually. This typically entails gathering information from hosts and visitors, and then making arrangements to satisfy preferences and requirements on both ends. Currently, there are few resources at the University to streamline this process, and whatever platform that does exist is limited to one specific student organization [1].

TigerNest: Event Organizer and Host Flow aims to resolve these issues by allowing visitors to sign up for rooms that Princeton hosts have volunteered to provide on the platform. This would leave the event organizer with the sole responsibility of "creating" the event and specifying event details. Moreover, the work required by visitors and hosts on the application would be more or less identical compared to the current process; in both scenarios, they are required to input basic identification information as well as preferences regarding their room type, though on TigerNest visitors will have the additional step of selecting the room once the visitor sign-up phase is in progress.

The development of this project can be broken down into four separate stages - requirements gathering, design, development, and evaluation. During the requirements gathering phase, the overall expectations of the application and additional caveats regarding the host matching process were determined based on conversations from representatives of various student organizations. Next, sketches and diagrams of the User Interface were designed to establish a rough idea of how the website will meet all of these requirements. The development phase consisted of coding the full-stack implementation of the event-organizer and host flows which were designed. Finally, after the application was deployed on a live server and tested by the developers, the representatives from the initial student organizations were asked to give feedback on the final product with specified instructions for navigating the platform.

2. Related Work

2.1 HackMIT

As mentioned previously, there is currently a platform to match hosts with overnight visitors for overnight accommodation, specifically for Hack Princeton, a hackathon hosted twice a year at the University that spans a total of three days [1]. Through this application, called "Localhost," hosts and visitors register on the website and are automatically matched with visitors via a max-flow algorithm [1]. This approach is based off of Hack MIT matching platform, though with some modifications, as will be discussed.

Max-Flow algorithms are a type of network flow algorithm, and are thus usually used in scenarios with directed graphs (collections of vertices and edges, where each edge has a specified direction) that have edge-capacities [2]. Each path through the graph, starting from a source vertex and ending at a sink vertex, is called a "flow," and each flow has its own capacity (similar to a pipe network where each pipe can handle a certain "flow" volume of water) [2]. The purpose of the max-flow algorithms such as Ford-Fulkerson and Edmonds-Karp is to maximize the total "volume" of the flow through the entire system while respecting each of the edge constraints [2].

Figure 1 depicts how the max-flow algorithm applies to the matching process for Hack MIT. The developers designate a “virtual source” directly connected to all the hosts, and a “virtual sink” directly connected to all the edges [2]. The edges from the source to the hosts represents their “host capacity,” or the maximum number of hackers they would be willing to accommodate.

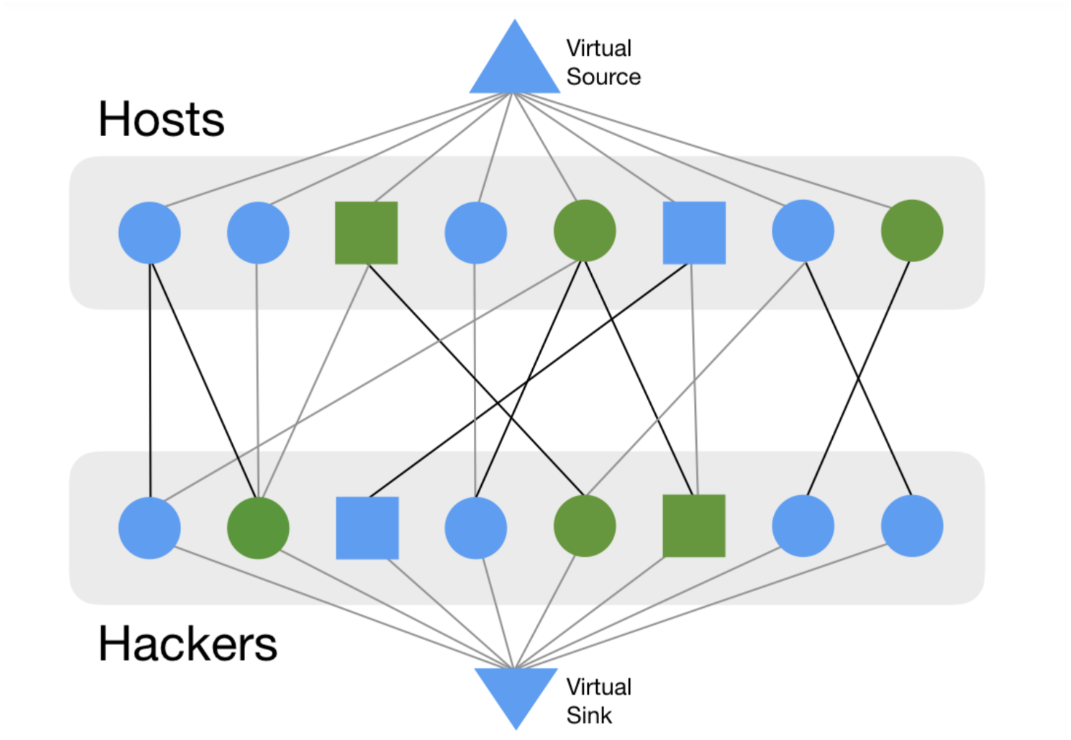


Figure 1: Hack MIT’s graph representation of max-flow matching between hosts and visitors

The graph in Figure 1 is a “bipartite graph,” meaning that all the vertices are among two disjoint sets - the Hacker set and Host set. The specific maxflow implementation used by the Hack MIT developers is the Edmonds-Karp algorithm [3].

The main advantage of the Hack MIT Edmonds-Karp implementation is that it ensures that all the Hackers are matched to a particular host, as opposed to a greedy algorithm that simply loops over each of the hackers and matches them to a host that is available [3]. Moreover, there is essentially no work done on the host or hacker side in the process besides signing up on the platform, as the pairings are all done automatically via the back-end algorithm.

2.2 Hack Princeton

As shown in Figure 2, Hack Princeton's Localhost uses the same general graph structure to represent pairings between Hosts and Hackers - a bipartite structure with a source vertex connected to the hosts, and sink vertex connected to the hackers [2]. However, in order to add an element of fairness in the matching process, ensuring that the Hackers are fairly evenly distributed among the Hosts, Localhost adds additional capacity to the hosts that can take on more visitors, and then uses a "best first search" algorithm, which is similar to the greedy algorithm except that decisions can be revised and updated [4], as opposed to Edmonds-Karp [2]. Moreover, Localhost also takes gender preferences into account by deleting hostings mid-algorithm if same-gendered rooms have visitors of both genders [2].

As with the Hack MIT approach, Hack Princeton's localhost has the advantages of ensured matching for all the hosts as well as minimal responsibility for hosts and hackers. Additionally, the pairings are fairly distributed among the different hosts throughout the process while also respecting gender preferences on both ends. However, one of the weaknesses of their approach that the organizers mentioned in an in person meeting is that the algorithm gives a preference to the visitors with the same-gender restriction by matching them first. This was done in order to ensure that visitors with the same-gender restriction are not left out during the process.

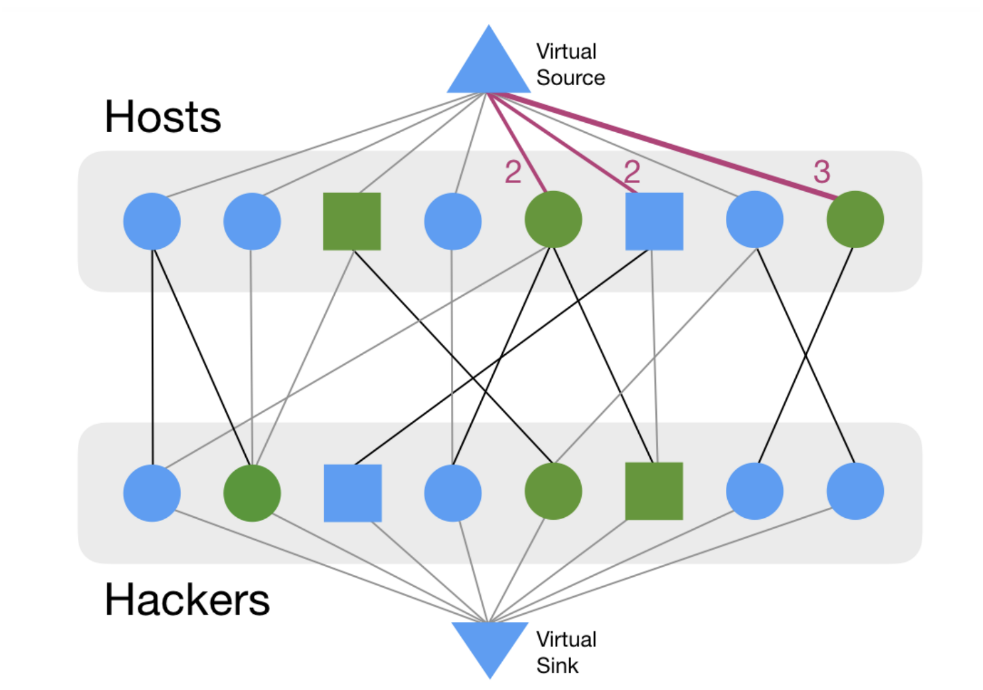


Figure 2: Hack Princeton's addition to MIT's graph representation, with increased edge capacity for hosts that can accommodate multiple hackers

2.3 Airbnb

Airbnb is an online marketplace used worldwide where people book properties or rooms that hosts have rent out.[1] Upon landing on the homepage, the guest specifies a specific date, the number of guests they will be having, and a particular city around the world. Based on this criteria, they are directed to a page will all the available rooms, with the prices of each displayed. The guest can further inspect a particular room based on pictures and additional notes provided

[1] "Airbnb." https://www.airbnb.com/. Accessed 6 May. 2019.

by the host, and then books the room and provides payment information. If one wishes to host, they navigate to the "become a host" section of the website and enter all the information about their place of stay that visitors should be aware of.

Part of the popularity of Airbnb is that it grants visitors the freedom to choose the place in which they are staying overnight, while also maintaining an extremely straightforward process for all types of users. The approach of TigerNest is inspired by this flexibility and simplicity Airbnb offers on its platform. Throughout the course of the project, it was ensured that the Event Organizers and Hosts sides of the application would have simple and intuitive flows.

3. Requirements Gathering

As mentioned in Section 1, the first and perhaps most significant stage of the project was the requirements gathering phase, where the heads of several student organizations were met to determine the overall structure of the application and the baseline features that it should implement. The student organizations were: Hack Princeton, Princeton Debate Panel, TigerLaunch, Princeton Envision, Club Swim, and Princeton Model United Nations, which are the primary organizations on campus that host overnight events. The project was also discussed with the Princeton Undergraduate Student Government (USG) TigerApps chair to determine what would be required for the project to be recognized as a TigerApp, as becoming a TigerApp will give the application a sense of legitimacy. Moreover, it was also suggested by Dr. Dondero to use the procedures outlined in *Interaction Design: Beyond Human-Computer Interaction* by Yvonne Rogers, Helen Sharp, Jennifer Preece while establishing requirements. As specified in Chapter 10 of *Interaction Design*, the three types of requirements that were gathered during this process were Functional Requirements, Data Requirements, and Environmental Requirements [5].

3.1 Functional Requirements

In terms of what actions the system should be able to perform, it was initially expected that we would need a platform where event organizers can create events, hosts can volunteer their rooms for those events, and visitors would sign up for their rooms. However, the organization heads also gave additional caveats about the host matching process that they felt the developers should be aware of. For instance, in many of these events, it is very common for hosts and visitors to drop out last minute, and thus the organization heads advised that we give users on all ends flexibility to drop out of the event last minute, or in the event organizers case, cancel the event altogether. Moreover, these organization heads also stated receiving emails in the last minute regarding changes in their room detail. For instance, a host may suddenly decide that they can take on more visitors after they discover their roommate will not be on campus for the weekend. An event organizer may also want to edit the location details or expected visitor account at some point during the process. Thus, it was suggested that having these edit features would beneficial to the users as well. Additionally, there needs to be some sort of data verification on the Event Organizer path that ensures that the student who logged in is indeed the head of an organization - in other words, a simple CAS authentication may not be enough, since a Princeton host may not necessarily be a student organization leader. The Hack Princeton organizers also mentioned their algorithm weighs visitors with same gender restrictions over visitors without those restrictions, and advised that the TigerNest implementation of the matching process avoids this bias, but at the same time ensure that same-gender visitors do not get left out in the process. Finally, many student organization heads recommended to leave payment options outside of the system, as subsidies and prizes for hosts are typically handled informally by members of the hosting organization.

3.2 Data Requirements

As with any other technical project handling data, there should be a steady, two-way flow of information between the back end and front end side of the application. In terms of how frequently the application would be used, there is not expected to be a large number of users on the site at a given point in time throughout the year, as there are around three or four overnight events being hosted each semester. However, some of these events, such as Hack Princeton and Princeton Debate Tournament, are large-scale, and thus the server should be able to robustly handle a large volume of hosts signing up for a given event at the same time.

3.3 Environment Requirements

The environment requirements, specifically, the technical environment requirements, were determined after meetings with the chair (head) of USG TigerApps Committee. According to the TigerApps Chair, TigerApps is requiring that all the applications it sponsors are developed with React.js, a front-end javascript framework, so that they can be more easily maintained. Additionally, the application must also be hosted on an AWS server in order to become a TigerApp, as this tends to be the most cost efficient service to host web applications and store data.[2] This applies both for applications that are under development, as well as applications that are currently hosted as TigerApps but with different frameworks - thus, this applications need to be shifted over to React.js and AWS in order to continue being hosted on the platform.

[2] "Amazon Web Services (AWS) - Cloud ...." https://aws.amazon.com/. Accessed 6 May. 2019.

4. Approach

4.1 Three Flows

In the previous section, three other platforms that pair guests with hosts were discussed. Among these three, Airbnb is the most closely related work to our platform, as TigerNest also expects visitors to select a particular room rather than automating the pairing process. Nevertheless, some aspects of Localhost and HackMIT application are incorporated in the project. Like Localhost, the host's gender preferences were collected to determine whether or not they were comfortable with members of the opposite sex staying in their room. Moreover, our platform also tries to prevent same-gender visitors from being left out by giving them a slight preference in the room selection process, but attempts to mitigate the extent of the bias that was present on Localhost (this was primarily handled in the implementation of *TigerNest: Visitor Flow*).

Based on the information collected during the requirements gathering phase, the most sensible approach was to break the application down into three separate "flows," or paths - the Event Organizer flow, Princeton Host flow, and overnight Visitor flow. Specifically, this project focuses on the design and implementation of the Event Organizer and Host flow, though all three flows are interconnected and share the same back end database.

Since the event organizers and hosts are required to be Princeton students, CAS authentication is required in order to progress in either of these flows. The event organizer branch of the application uses an additional authentication layer to verify that the student is indeed the head of an organization, as mentioned in Section 3.1 - if the student's Princeton netid is not recognized by the system, they are directed to a registration page that requires a "registration code," which can only be obtained by meeting with the students managing the website (Niranjan Shankar and Michelle Yuen, or the TigerApps chair if the application is to be approved). Upon verifying that the student is indeed an organization head, they will be given the registration code and can proceed through the entire event organizer path.

In order to satisfy the USG TigerApp requirements, it was decided that the front-end would be developed with the React.js front-end framework, hosted on an Express.js server (part of the Node.js javascript javascript server framework commonly used with React).[3] As there was not a particular requirement regarding the back-end, the application uses PostgreSQL due to its compatibility with various deployment software, such as Heroku[4] and Amazon RDS,[5] and since a relational database format was the most appropriate for representing the data for event organizer, host, events, and matchings.

In terms of deployment, the initial goal that was established was deployment to an AWS server. However, upon investigating other applications at Princeton, it was decided that deployment to Heroku should be the primary goal of the application, and deployment to AWS should be labeled as a "stretch" or "reach" goal.

4.2 User Interface Design

As stated in Section 1, before the actual development of the application, the user interface for the entire event organizer and host flows was designed on pencil and paper sketches. Each possible scenario in both paths were brainstormed. The designs were helpful in ensuring that the

[3] "Create a New React App – React." https://reactjs.org/docs/create-a-new-react-app.html. Accessed 6 May. 2019.
[4] "Heroku Postgres." https://www.heroku.com/postgres. Accessed 6 May. 2019.
[5] "Amazon RDS for PostgreSQL – Amazon ...." https://aws.amazon.com/rds/postgresql/. Accessed 6 May. 2019.

developed application would be able to carry out all of the core functionality features while still maintaining a simple and intuitive flow. Figure 3 represents an example of a sketch for one of the pages on the Event Organizer path.

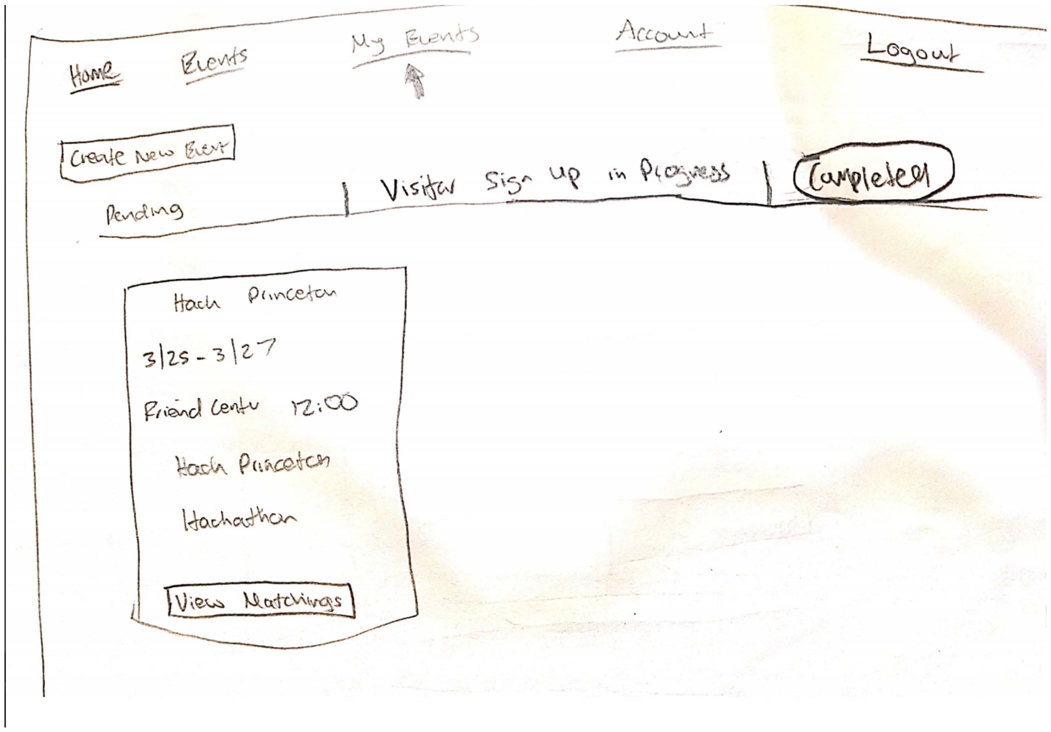


Figure 3: A sketch of the “My Events” page on the Event Organizer Path from the User Interface Design Phase, showing an event in the “Completed” phase

5. Implementation

5.1 Breakdown

The web application was developed as a team composed of two individuals - myself, Niranjan Shankar, and Michelle Yuen, whose projects were TigerNest: Event Organizer and Host Flow, and TigerNest: Visitor Flow, respectively.

Each of us were responsible for the full-stack implementation of each of our respective paths, all from the database layer, to the servers, and the client side. Endpoints were added to the same database API by the individual using that particular endpoint. Since the visitor-flow server did not use CAS Authentication, the server for the event organizer and host path was kept separate from the server handling the pages on the visitor flow. Each of us also separately developed the frontend pages that were specific to our paths.

5.2 System Architecture

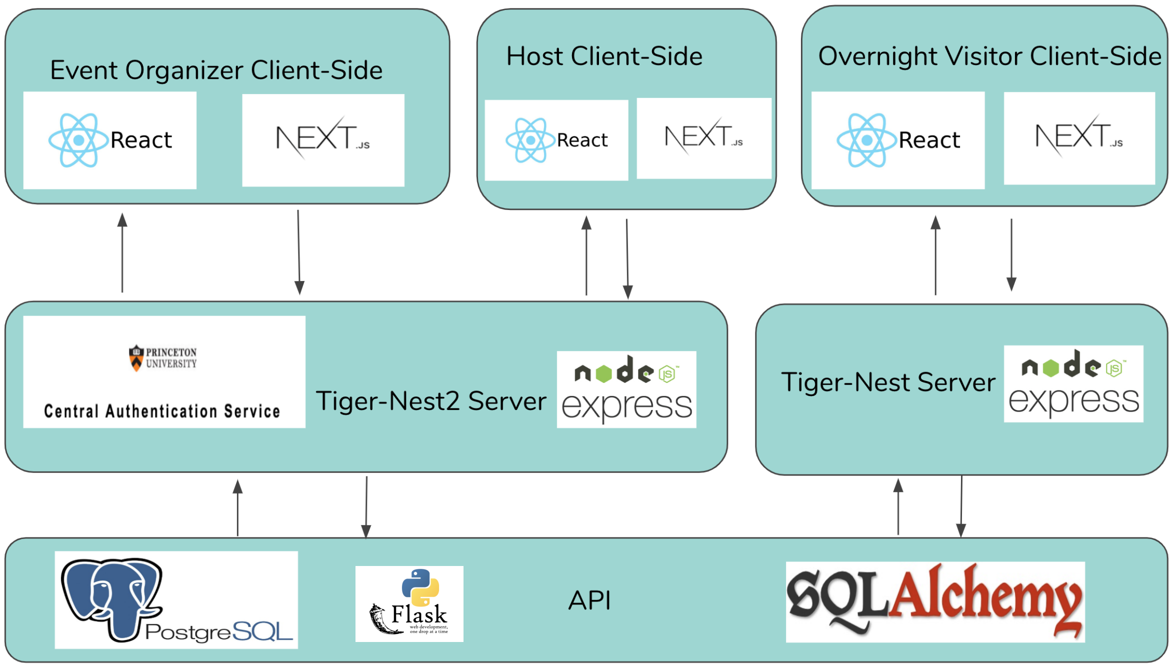


Figure 4: System Architecture

Figure 4 represents the overall structure of the entire application. The aspects that were pertinent to *TigerNest: Event Organizer and Host Flow* were the API, Tiger-Nest2 server, and the Event Organizer and Host client sides. The database was hosted on a PostgreSQL server with

Python Flask, and was queried using SQLAlchemy, a Python SQL toolkit.[6] The Tiger-Nest2 express server facilitates GET and POST requests between the database and the frontend, and also handles all the pages on the event organizer frontend, while also ensuring that the user is authenticated via the Princeton Central Authentication Service. The Event Organizer Client Side and Host Client Side both use React.js for displaying the User Interface, as well as Next.js, a JavaScript framework used to handle routing between pages on the frontend that is often used in conjunction with React.js[7].

5.3 Front-End

As summarized in Section 5.2, the User Interface of the application was designed using React.js and Next.js. Typically, a React.js application is broken down into "Components" and "Pages," with each page consisting of one or several components, as well as additional functions for handling those components. For instance, the navigation bars were organized as a component in the nav.js and hostNav.js files, and imported on the pages displaying them.

Other pertinent libraries used throughout the front-end development were ReactStrap, a library that provides the developers with Bootstrap components to organize the interface[8], and helped make the User Interface more aesthetic and navigable. A common theme throughout the client side is representing the "events" as the ReactStrap Card Component, and asking the user to specify information via the ReactStrap Modal. Figure 5 gives a visual example of this theme.

The js-Cookie library was used to send information regarding the organizer's netid from the express server to the React frontend, which then ensure that the events and hostings being displayed on the platform were unique to that Event Organizer and Host. Moreover, for the Event

[6] "SQLAlchemy - The Database Toolkit for Python." https://www.sqlalchemy.org/. Accessed 6 May. 2019.
[7] "Next.js." https://nextjs.org/. Accessed 6 May. 2019.
[8] "Reactstrap." https://reactstrap.github.io/. Accessed 6 May. 2019.

Organizer path, the netid cookie was also used to verify whether or system recognizes the user as a student leader that was approved by the system managers.

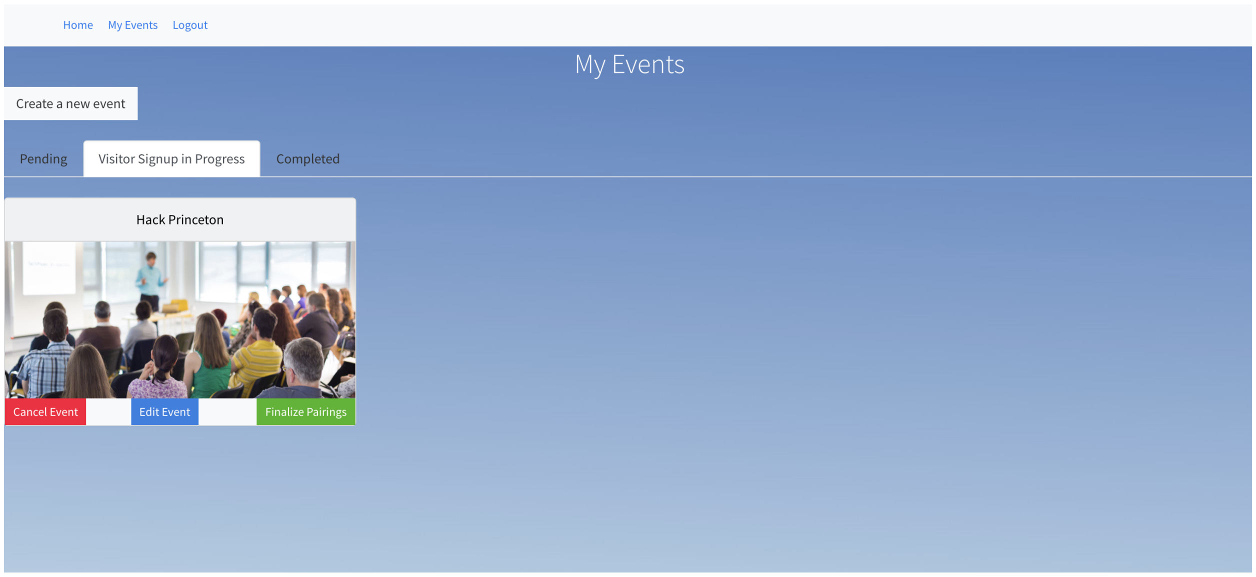


Figure 5: The developed "My Events" page of the event organizer flow, with the "Hack-Princeton" event organized as a card, and tabbing to distinguish between event types.

5.4 Tiger-Nest 2 Server

The Tiger-Nest 2 server is hosted on an express server, and receives all the various endpoints that may be hit on the Event Organizer and Host sides of the application. The express server facilitates the requests between the client side and the database, while also validating the user as a Princeton student via Central Authentication Service (CAS). The express library used to connect the application to Princeton's CAS framework is 'cas'.

The CAS Verification procedure is as follows: a "catch-all" endpoint first redirects to a "verify" method within the express server file. This method calls a "validate" method using the 'cas' library, and assesses whether or not the cas ticket is valid given the user's credentials. If the login is successful, the user is redirected to the home page.

5.5 Database

The relational database that was used on the back-end was managed with Flask, SQLAlchemy, and PostgreSQL. The relevant tables of the database for the event organizer and host paths are as follows:

- Event Organizer (the organization heads that create and manage event objects)
- Event (The event that will be hosted, that hosts and visitors can sign up for)
- Host (General information about the host)
- Pairing (Represents a "room" that was volunteered by a host for a particular event - contains all the information about the host preferences).

5.6 Deployment

The baseline goal for the deployment of the application was to a heroku server, which was successfully executed. Separate heroku accounts were created both for the back-end and the front-end of the application. The backend is hosted at https://tigernest-backend.herokuapp.com, whereas the front-end of the TigerNest2 server is hosted at https://tiger-nest2.herokuapp.com (though the initial home page can be found on the https://tiger-nest.herokuapp.com server). Heroku databases are generally accepted via one of three management platforms: PostgreSQL, Redis, and Apache Kafka, which is among the reasons that PostgreSQL was selected for our backend implementation. For a cleaner deployment procedure, two additional repositories were created - one designated with front-end deployment, another designated with back-end deployment. This way, the any errors on the console logs could be more easily distinguished between errors on the front-end and errors on the back-end.

5.7 Challenges

Perhaps the most challenging obstacle in the development of the application was CAS Authentication. Currently, most of the resources provided at Princeton regarding CAS validation are for applications that use python frameworks like Flask and Bottle. Virtually no help was available through professors or Princeton courses for integrating CAS in an application that uses frameworks like React and Express. Thus, the first attempts to add this feature to the application were based on google tutorials, but were unfortunately unsuccessful.

Thankfully, another Princeton student suggested meeting the developer of Princeton Courses, Mel Shu, to address this dilemma. After two separate meetings (over the span of two weeks) with Mel, CAS was successfully integrated in our application, with reliable exchange of cookie-session information from the server side to the front end. Nevertheless, this still drained much precious time that could have been used to develop other features of the application, and potentially to meet some of the stretch goals that were abandoned.

Another difficulty encountered over the course of the project was attempting to deploy the project to Amazon Web Services. After a successful deployment of an early version of the project to Heroku, an attempt to deploy the project to AWS was made, initially through an instance on Elastic Beanstalk, which is a widely used AWS Framework to deploy web applications[9]. Unfortunately, this was met with various errors that were rather cumbersome to resolve, such as the formatting of "build" statements and configuration with various AWS parameters. The next attempt was on an AWS S3, instance, which is a storage software manager on AWS[10]. Though a static html page was successfully deployed to the s3 instance that was

[9] "AWS Elastic Beanstalk – Deploy Web Applications - Amazon.com." https://aws.amazon.com/elasticbeanstalk/. Accessed 7 May. 2019.
[10] "Amazon S3 - AWS - Amazon.com." https://aws.amazon.com/s3/. Accessed 7 May. 2019.

created, getting the entire application compatible with s3 would have required restructuring much the code and the repository, since server side rendering on s3 tends to be rather complicated with dynamic rendering with servers. Thus, after a week and a half of experimentations with AWS, it was decided that it would be in the best interest of the project to temporarily abandon this goal and shift attention on the core functionality features. However, the main goal of deployment to Heroku was successfully executed, and thus the final version of the platform is still available for live use by the Princeton community.

It was also worth noting that learning the front end frameworks involved to develop the application was an initial hurdle. The developer, myself, was not familiar with the typical structure of React and Express, and thus the first stages of the development phases were rather slow in terms of progress and efficiency. Moreover, two entire full-stack flows over the span of the semester was certainly not an easy endeavor to tackle. Nevertheless, after these initial hurdles were crossed, the development phase started to progress faster.

6. Evaluation

First, the event organizer and host paths of the application were tested and evaluated by myself, in order to ensure that all the core implementation features were functional and running smoothly. After addressing various bugs that were detected after deploying the newly updated code, a MoSCoW assessment was conducted by myself in order to assess whether the application hits all the functions that were designated as high priority during the requirements gathering phase.

6.1 MoSCoW Assessment

The MoSCoW template allows developers to prioritize different features of the application into Must Have, Should Have, Could Have, and Won't Have Features. With respect to *TigerNest: Event Organizer and Host Flow*, here are the broken down priorities of the various features:

- Must Have:
    - "Event creation" functions for Event Organizers
    - "Sign up to Host" functions for Hosts
    - Some method of authenticating users with a username-password system.
    - Personalized session - storage for event organizers and hosts.
    - Deployment to a live server
    - Two-Way flow of data between both ends.
    - Hosts must be able to view visitor information
- Should Have:
    - Options to edit and delete hosting or event information
    - Authentication via Princeton Central Authentication Service
    - Event organizers can view pairing information between hosts and visitors
    - Smooth routing between pages
    - An satisfactory User Interface.
- Could Have:
    - Deployment to an Amazon Web Services Server
    - Functions for hosts to sort for events between specific dates

    - Event Organizers can automatically send email notifications to hosts and visitors through the platform.
    - Professional-looking User Interface
    - Options to have an automated pairing process if the event organizer wishes.
    - Image uploading options for the hosts and visitors.
- Won't have:
    - Will not handle payments from event organizers to hosts

Based on these outlines, it appears that *TigerNest: Event Organizer and Host Flow* meets all of the core "must have" features as well as the "should have" features. The system can be used by event organizers to organize overnight events, and by Princeton hosts to volunteer their rooms for overnight accommodation for the visitors attending these events.

Needless to say, the experience of users on the platforms could certainly be enhanced if the could have features, such as a more professional looking UI and options to sort and filter based on the start dates for the event. It would also be convenient if event organizers had an option to send out emails to mass listservs with the click of a button. However, as these are only "could have" priority level features, they do not take away from the main functionality and the primary goals of the system.

6.2 External User Evaluation

An external user from one of the potential client organizations, Princeton Debate Panel, was asked to perform a series of tasks and offer feedback while navigating the platform. The list of tasks is described in the following section. Questions and comments that the user made were recorded and considered. These comments regarded logic bugs in the application, potential improvements to the User Interface, and additional functionalities that could be set in place.

- After reaching home page, the user tried entering the host path of the application before receiving the instruction to attempt the Event Organizer Path first. It was discovered that logging into CAS via the host path first inadvertently directs to a page on the Event Organizer Path. This was recorded as a bug.
- The user entered the event organizer aspect of the platform and was redirected to the Event Organizer Registration page. The user asked for the purpose of the page, and upon hearing the explanation, agreed that this was a useful mechanism for distinguishing between Princeton students and organization heads.
- After entering the registration code, the user was then able to enter the “My Events” page, and almost immediately clicked the “Create an Event” button. This suggested that they understood the overall flow of the page and found the page intuitive.
- The user was asked to submit the form without entering in all the mandatory inputs, and to confirm that an alert window popped up. This confirmation was positive.
- The user created their event, with a name of their choice, and then asked if they should put the event in the Visitor Sign-Up phase, and further requested an explanation of what the visitor sign-up phase is. It was explained to them that this would put the event in a stage in which the visitors are allowed to sign up for the event. The user found it helpful that the event phases were distinguished.
- After putting the event in the visitor sign-up phase, the user asked how to sign up to host for the event that they created. They were instructed to click the “home” button and

navigate to the Host path from there. However, the user suggested for efficiency's sake that it might be a better option to have a hosting option on the event organizer path itself.

- The user landed on the host all events phase, and was asked to confirm that they were able to see the event they had just created. This confirmation was positive.
- The user proceeded to sign up for the event that they had created. They were asked once again to submit the form without inputting all fields, and confirm that an alert message popped up on the screen. This confirmation was positive.
- After submitting the host form, the user asked how the visitors would be matched with the hosts, and how the algorithm would take gender preferences into account. It was explained to them that the visitors would view the available rooms and choose their accommodation for the night, like they would with "Airbnb." The user complimented this approach and expressed that this was a creative way of tackling the problem.
- The user asked why the event they signed up for disappeared on the host "All Events" page. Upon hearing that their event can be found on the "My Events" page, and no longer on the "All Events" page, the user suggested that the All Events page should show every event regardless, but display the host's specific events slightly differently. For instance, with an "Already signed up" notification on the Card.
- The user was asked to navigate to the My Events page, where they were asked to complete the last step, which was to edit the number of visitors they were willing to take and click "Update Info." Upon clicking this, the user confirmed that the my events display indeed had the newly updated number of visitors.

- The user was finally asked if they had any further recommendations, and if the platform provided more or less what they would expect from a host-visitor pairing application. They offered the following feedback:
  - Photo uploads by the hosts so that visitors will be able to recognize them upon meeting them.
  - Some sort of location or map API so that the visitors will be able to find the dorm in which they were staying in (this would primarily be relevant to the Visitor Flow, but this was recorded and considered nonetheless).
- The user was thanked for their time, and to let the developers know if they had any further suggestions.

This assessment was helpful in identifying bugs that would have otherwise been difficult to catch through one evaluator alone. Moreover, it confirmed that the must have and should have features of such an application were met, and that the overall "Airbnb" approach will win the approval of potential clients. The user's comments also suggested platform is ready to be used to members of organizations that host overnight events, though the potential modifications that they offered would make TigerNest even more convenient, and thus more likely to be used down the road, by the clients.

7. Conclusion

Overall, *TigerNest: Event Organizer and Host Flow* succeeds in its fundamental goal to reduce the burden of organization leaders in terms of managing overnight accommodation. The User Interface is fairly navigable and intuitive, as validated by the evaluation feedback, and there is a steady and reliable flow of data between all the layers of the application. Additionally, all of the high priority requirements established during the requirements gathering phase were met, and the system was rigorously tested to ensure that it meets these expectations.

7.1 Takeaways

The project was a learning experience not only because of the new languages and frameworks used, such as React.js, Express.js, and the AWS servers that were attempted, but also because of the requirements gathering and evaluation phases, which were integral to the project. These stages separated *TigerNest: Event Organizer and Host Flow* from traditional coding projects for university classes. This ability to gather requirements through meetings with the application's clients is a must-have in the repertoire of a successful software engineer. Additionally, learning how to incorporate these suggestions to improve the overall user experience is essential for maintaining high quality software applications. The project also helped reinforce how models such as MoSCoW are pertinent to the development life cycle. Another noteworthy takeaway from the project was the utility of User Interface sketches. Planning out the entire logical path through a particular workflow on the application certainly expedited the actual development phase.

7.2 Future Developments

One potential "could" have feature of the application that will be looked into is another attempt at deploying TigerNest on an AWS server, and potential sponsorship as a TigerApp. Two of the three requirements specified by the TigerApps chair was met: the front end was developed using React.js, and can thus be more easily maintained by TigerApps members, and the application idea and approach was approved by the TigerApps chair. All that is left in terms of conditions is that the application is hosted on an AWS server. Though the initial attempts at deploying to AWS were unsuccessful, at this point it would be possible to devote all the attention solely to AWS deployment without worrying about pending core features. Fortunately, there is another AWS platform, namely Elastic Compute Cloud, that is a popular resource for web developers that may be worth looking into[11].

Additionally, the could have features as well as the recommendations from the user during the evaluation phase, as detailed in Section 6, will be considered in order to enhance the application. For instance, options such as a “search bar,” and image uploading for the hosts upon registration, can indeed be helpful for the organizers and hosts. Additionally, keeping in mind some of the concerns expressed by the user during the evaluation, additional viewers can be asked to review our application and gather their feedback, and thus determine the individual priority of each of the new features that will be implemented on the platform. Though the single user evaluation was useful, it would certainly be helpful to gather more opinions as well from other clients.

8. Acknowledgements

TigerNest: Event Organizer and Host Flow would not have been possible without my advisor, Dr. Robert Dondero, who helped me stay on track to completing the application over the course of the semester, and offered advice regarding the priority of various features, people to be contacted to overcome roadblocks, and the most logical steps forward. I would also like to thank my teammate Michelle Yuen, who worked on the visitor flow of the project in *TigerNest: Visitor Flow*. Additionally, without Mel Shu, the developer of Princeton Courses, it would have been nearly impossible to integrate CAS in our system. Finally, the advice from the members of the

[11] "Amazon EC2 - AWS - Amazon.com." https://aws.amazon.com/ec2/. Accessed 7 May. 2019.

student organizations that agreed to discuss the application throughout the requirements gathering and evaluation phases was invaluable in developing and assessing the platform:

- An Lahn Le, Arjun Sai Krishnan - Princeton Debate Panel
- Baran Cimen - Princeton Envision
- Bozhidar Stankovickj - Princeton Model United Nations
- Stephen Cornwell - TigerLaunch
- Casey Chow, David Fan, Gordon Chu - Hack Princeton
- Yannis Karikozis - Club Swim
- Reilly Bova - TigerApps

References

[1] Chow, Casey. “Lessons Learned from HackPrinceton's Host Matching (Part 1).” *Medium*, HackPrinceton, 5 Dec. 2017, medium.com/hackprinceton/lessons-learned-from-hackprincetons-host-matching-part-1-ad3cf1ef7e24.

[2] Sedgewick, Robert, and Kevin Daniel Wayne. *Algorithms*. W. Ross MacDonald School Resource Services Library, 2017.

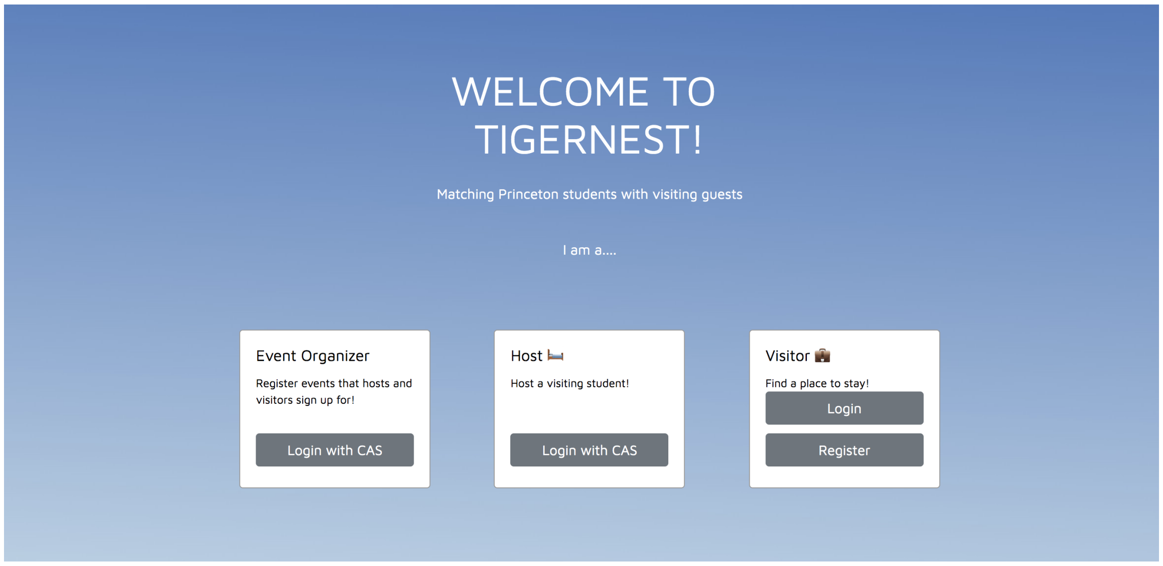
[3] Athalye, Anish. “Algorithms in the Real World: Host Matching.” *Medium*, HackMIT Stories, 25 Sept. 2015, medium.com/hackmit-stories/algorithms-in-the-real-world-host-matching-62baf4d7c165.

[4] Singh, Shambavi. “Best First Search (Informed Search).” *GeeksforGeeks*, 31 Jan. 2019, www.geeksforgeeks.org/best-first-search-informed-search/.

[5] SHARP. *INTERACTION DESIGN: beyond Human-Computer Interaction*. JOHN WILEY & Sons, 2019.

Appendix

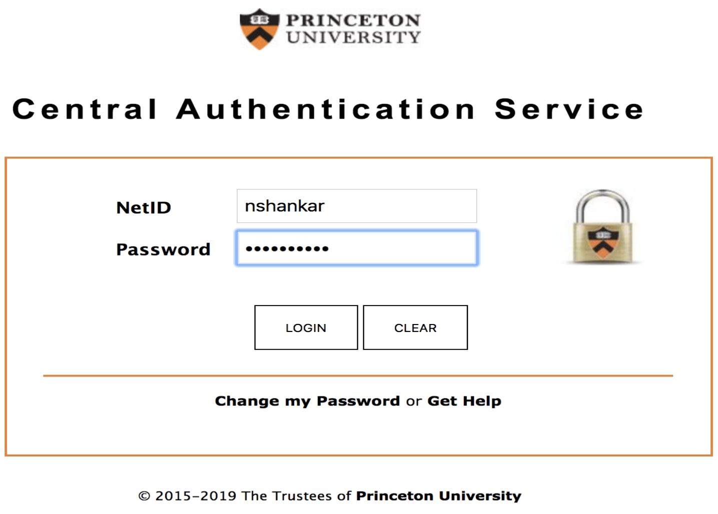
(What it does Walkthrough):


WELCOME TO TIGERNEST!

Matching Princeton students with visiting guests

I am a....

1) The user navigates to https://tiger-nest.herokuapp.com, and clicks the "login with CAS" on the Event Organizer card.

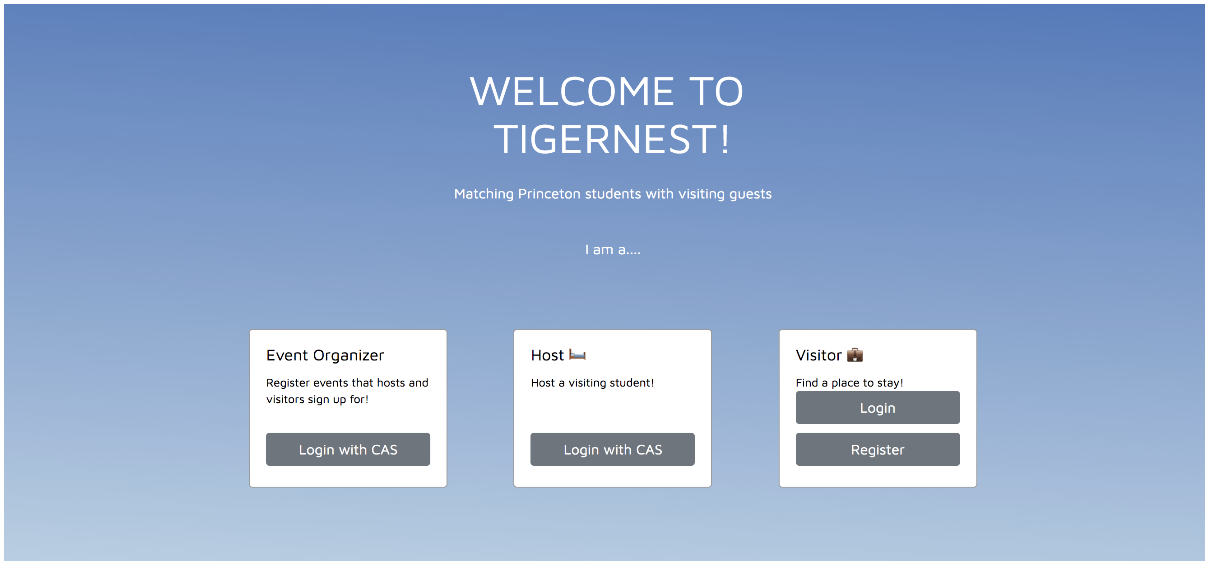

Central Authentication Service

NetID: nshankar

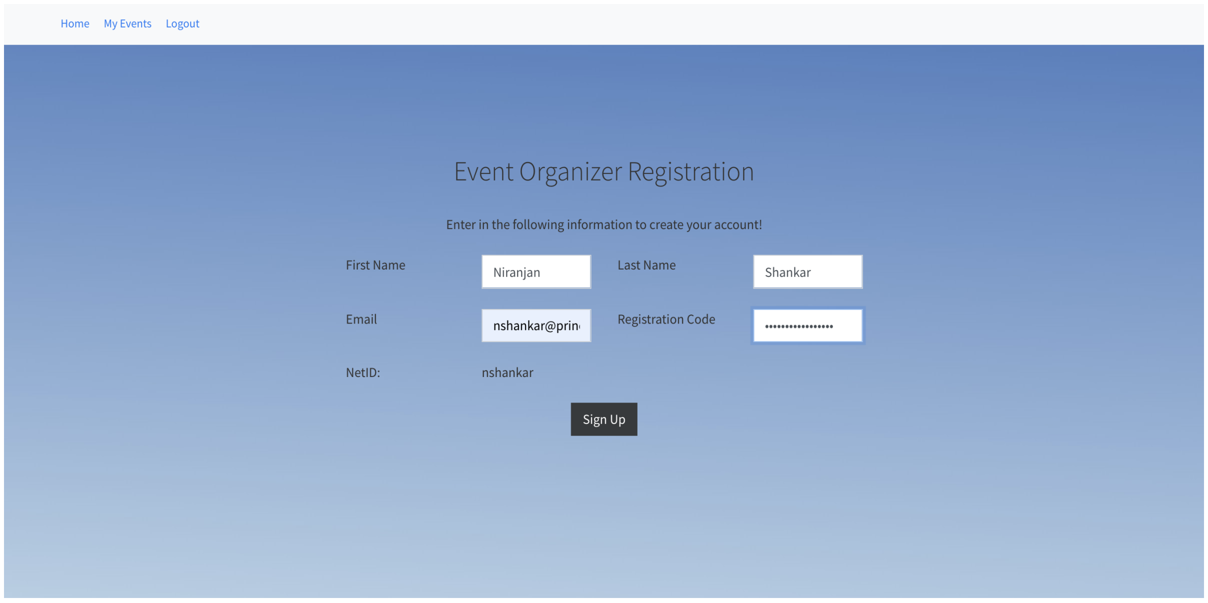
Password: ••••••••••

© 2015–2019 The Trustees of Princeton University

2) The user enters their login credentials with CAS

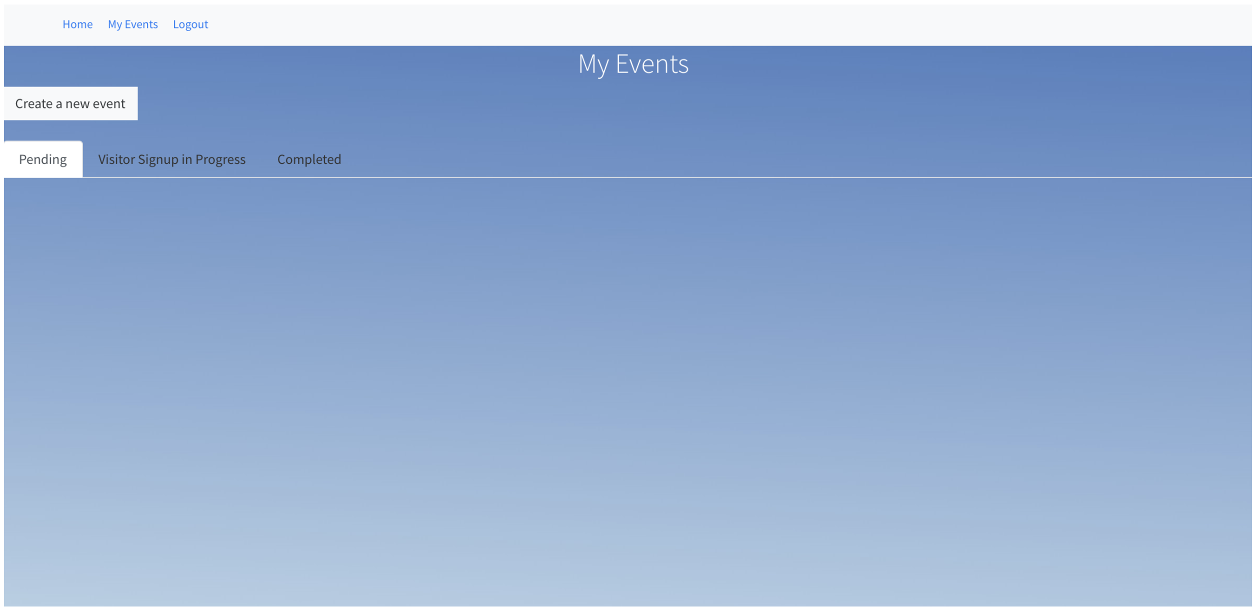


3) The user clicks “Login with CAS” on the event organizer card again

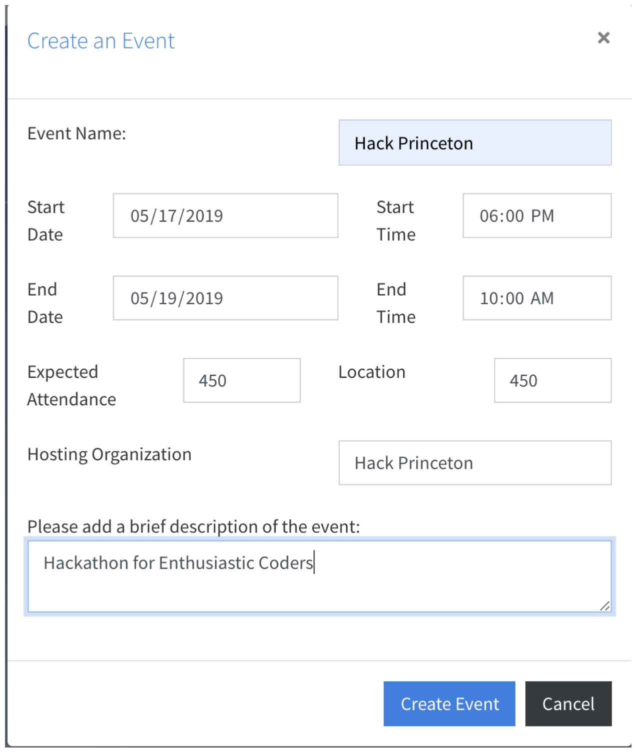

Home
My Events
Logout
Event Organizer Registration
Enter in the following information to create your account!
Sign Up

First Name	Niranjan	Last Name	Shankar
Email	nshankar@prin	Registration Code	••••••••••••••••
NetID:	nshankar		

4) The user is direct to an “Event Organizer Registration Page.” For the first name tab, they will enter their first name. In the last name tab, they will enter their last name. For the email input, the user will enter in any personal email address of their choice. For the registration code, the user must input the following: 2sdi319230d8208sd.



5) The user is directed to the My Events page. They will click the "Create a new event" button.



Event Name: Hack Princeton

Start Date 05/17/2019 Start Time 06:00 PM

End Date 05/19/2019 End Time 10:00 AM

Expected Attendance 450 Location 450

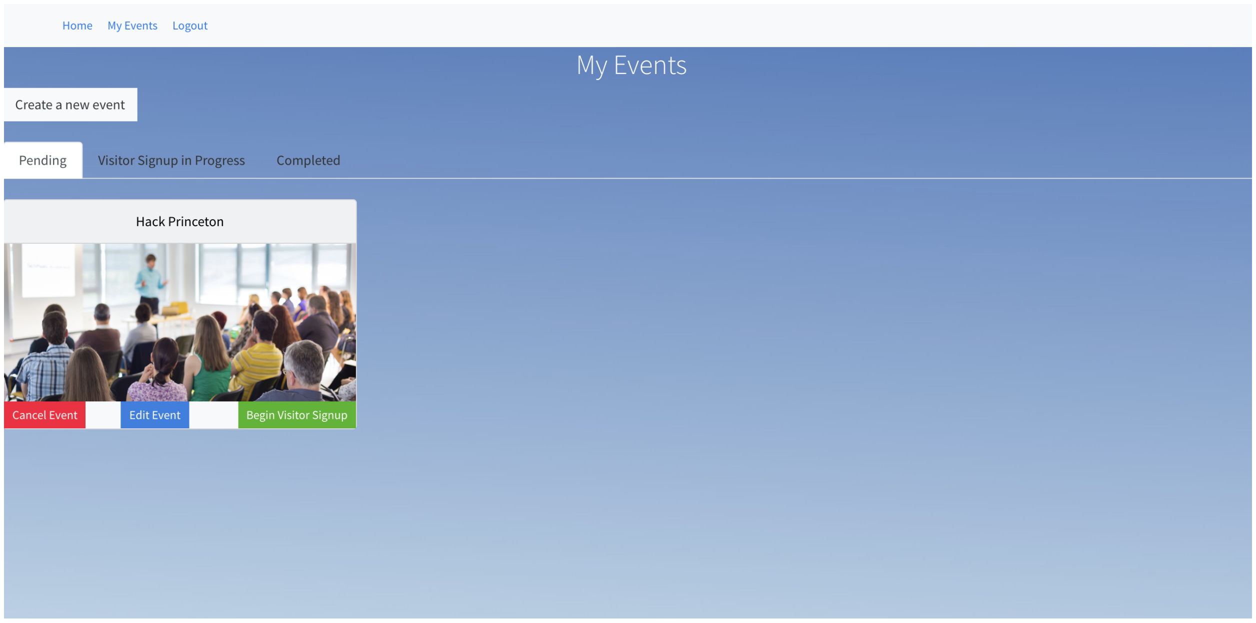
Hosting Organization Hack Princeton

Please add a brief description of the event:

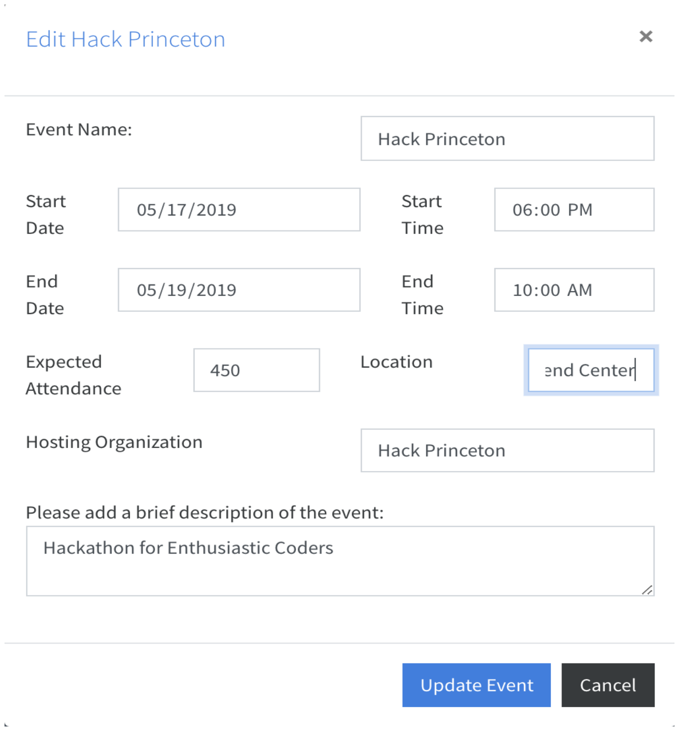
Hackathon for Enthusiastic Coders

6) The user is prompted with a "Create an Event" Modal, where they will input the following information: Hack Princeton for the event name, 5/17/2019 for the start-date, 6:00 PM

for the start time, 5/17/2019 for the end-date, 10:00 AM for the end time, 450 for the expected attendance, 450 for the location, Hack Princeton as the hosting organization, and "Hackathon for Enthusiastic Coders" as the description. Then, the user will click "Create Event."



7) The page will automatically refresh and the user should see the newly created "Hack Princeton" event under the "Pending" tab.


Edit Hack Princeton

Event Name: Hack Princeton

Start Date: 05/17/2019 Start Time: 06:00 PM

End Date: 05/19/2019 End Time: 10:00 AM

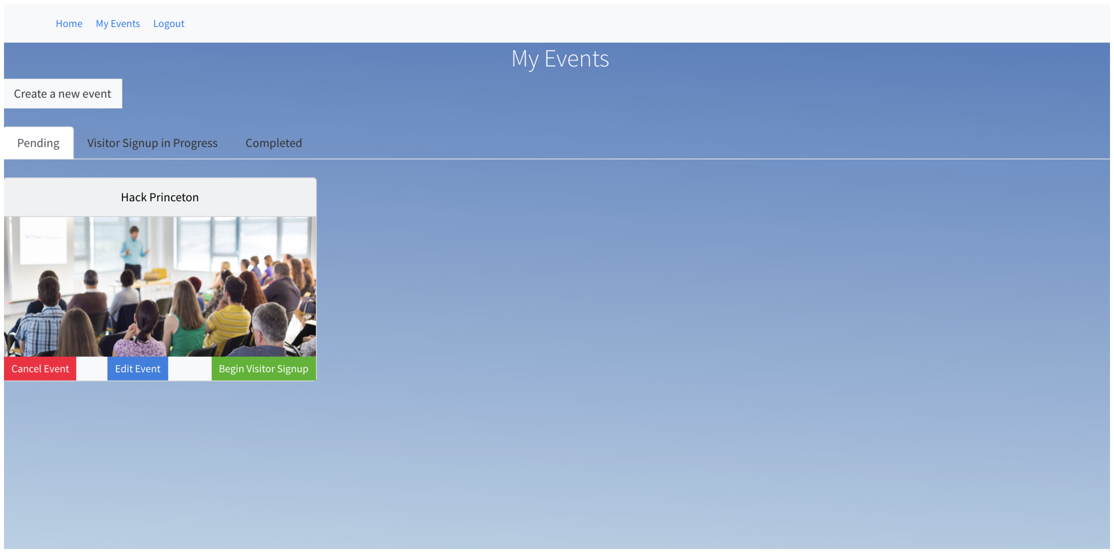
Expected Attendance: 450 Location: ɘnd Center

Hosting Organization: Hack Princeton

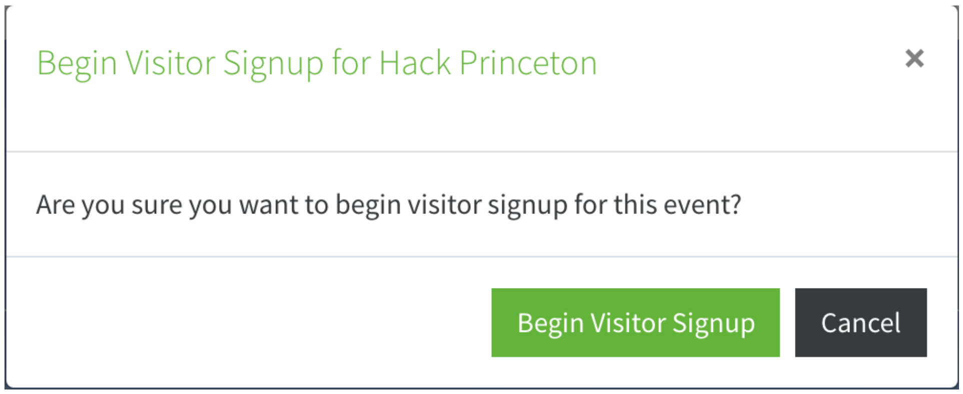
Please add a brief description of the event:

Hackathon for Enthusiastic Coders

8) The user will then be prompted with the “Edit Hack Princeton” Modal. Under the location field, they will change the value from 450 to “Friend Center,” and then click Update Event.



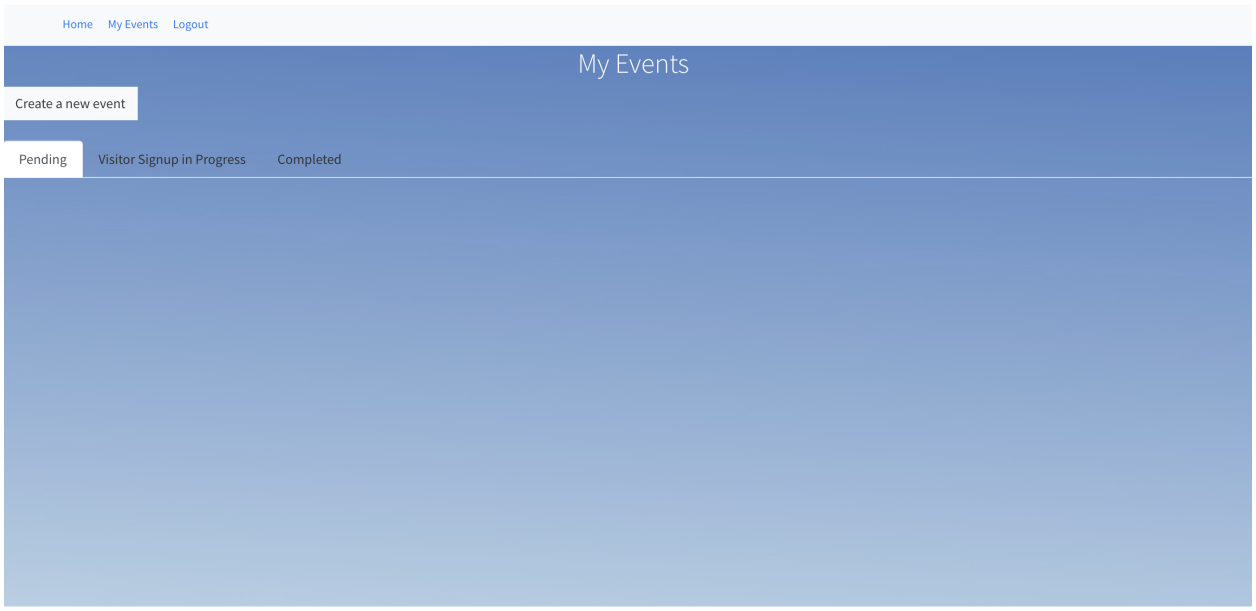
9) The event will be visible again under the “Pending Tab.” Now, the user will click, the Begin Visitor Signup button.



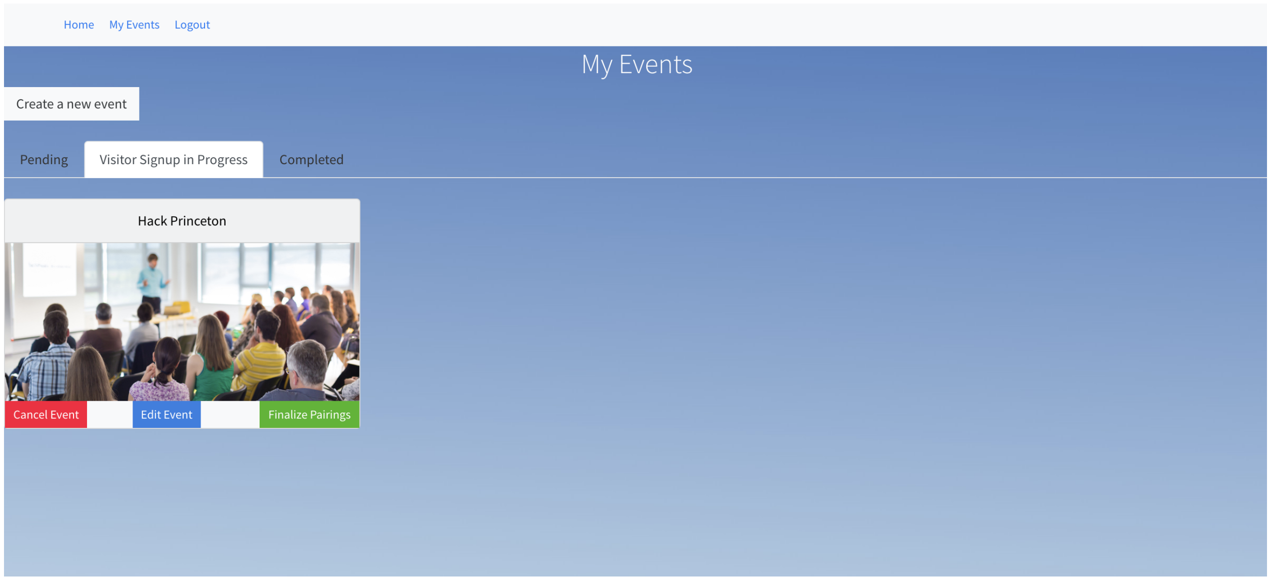
Begin Visitor Signup for Hack Princeton

Are you sure you want to begin visitor signup for this event?

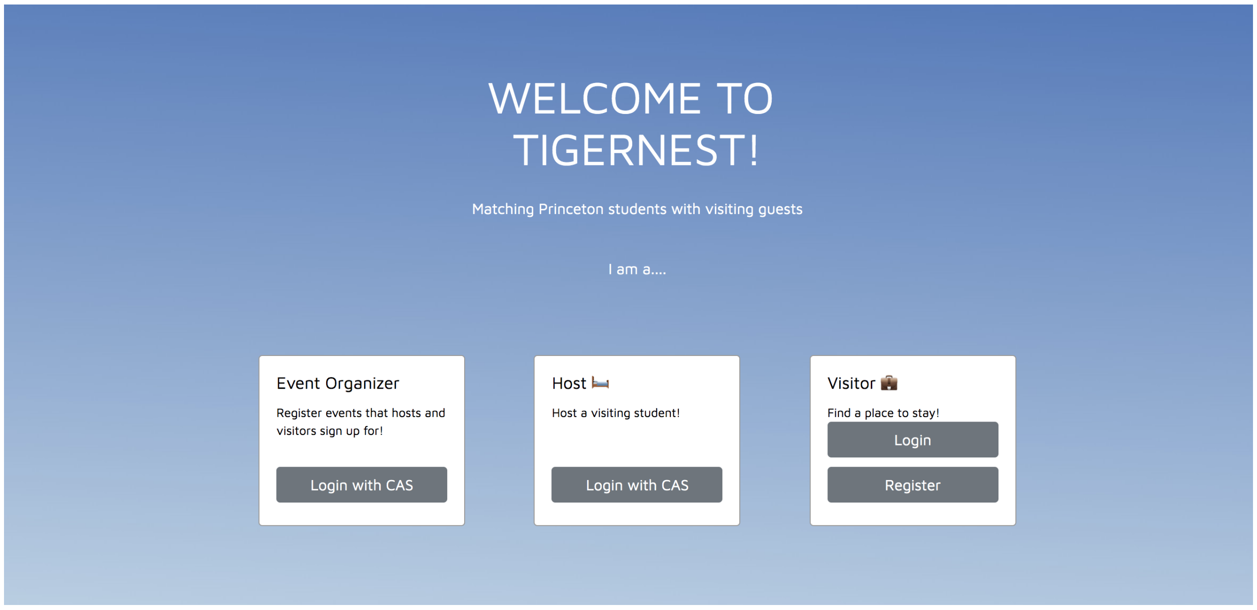
10) The user will select “Begin Visitor Signup” once on this Modal.



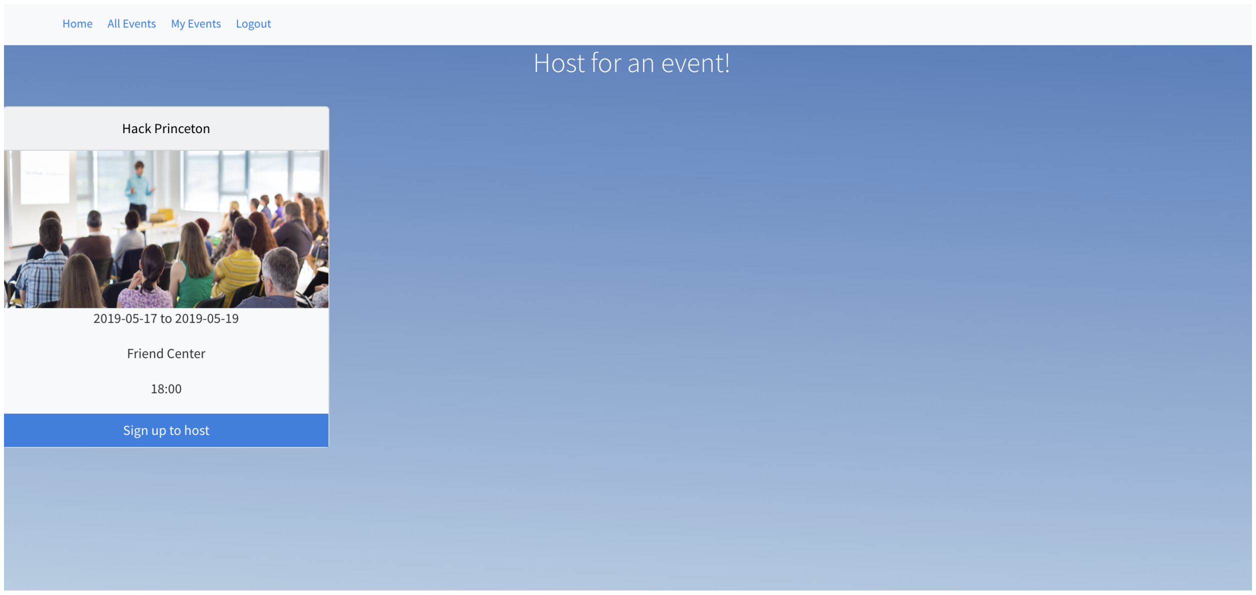
11) The page again will refresh, and the user will no longer see the Hack Princeton Event under the Pending Tab. The user will then click the "Visitor Signup in Progress" tab.



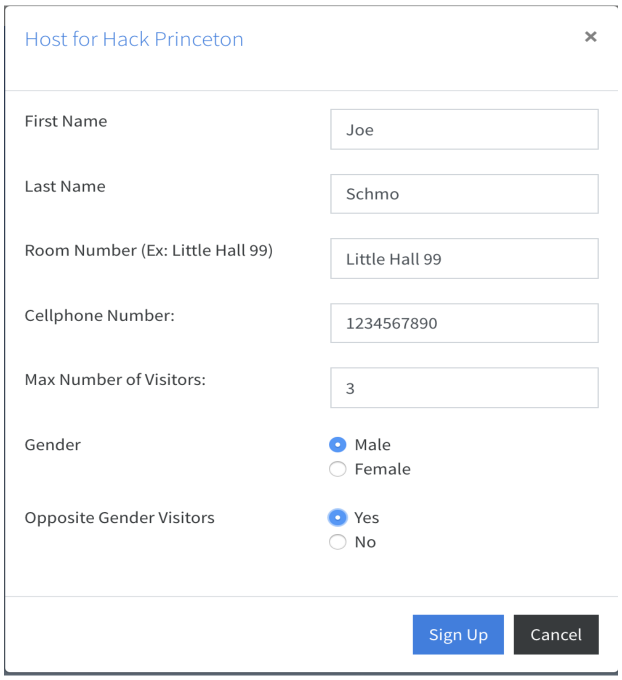
12) The user should now be able to see the Hack Princeton event under the Visitor Signup in Progress Tab. Next, the user should select the "Home" key on the Navigation Bar at the top of the page.



13) The user will be directed to the home page again, where they should click "Login with CAS on the host card.

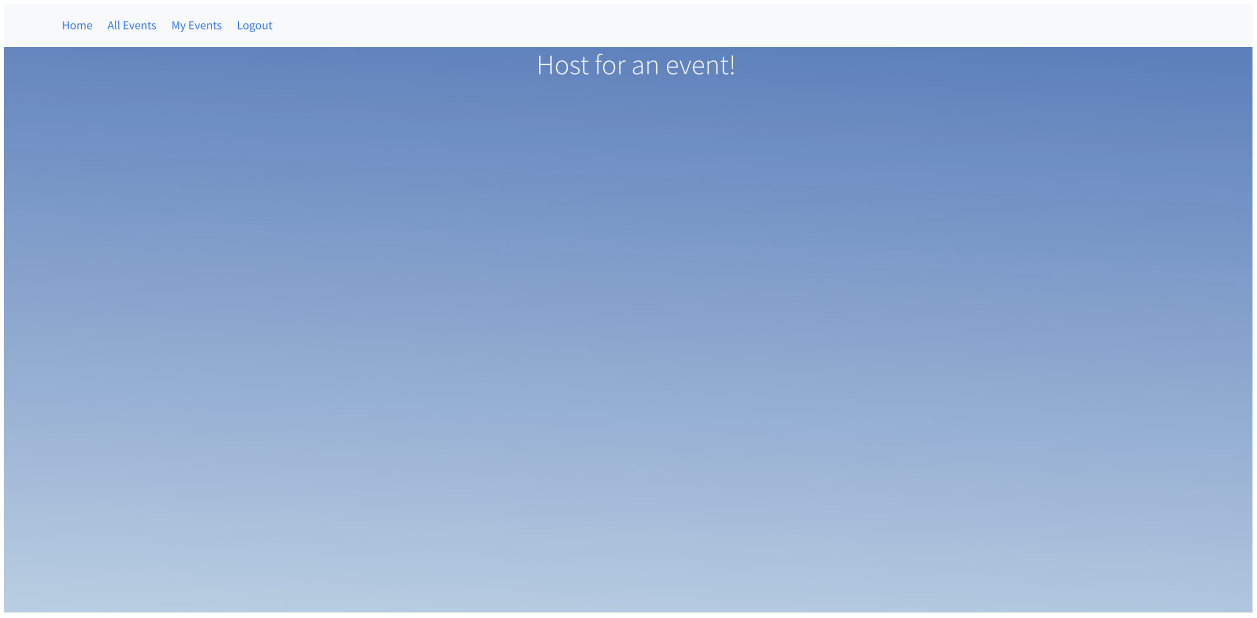


14) The user will arrive at the "All Events" tab on the Host path, where they should be able to view the event that they created as an Event Organizer. They will click "Sign up to host"

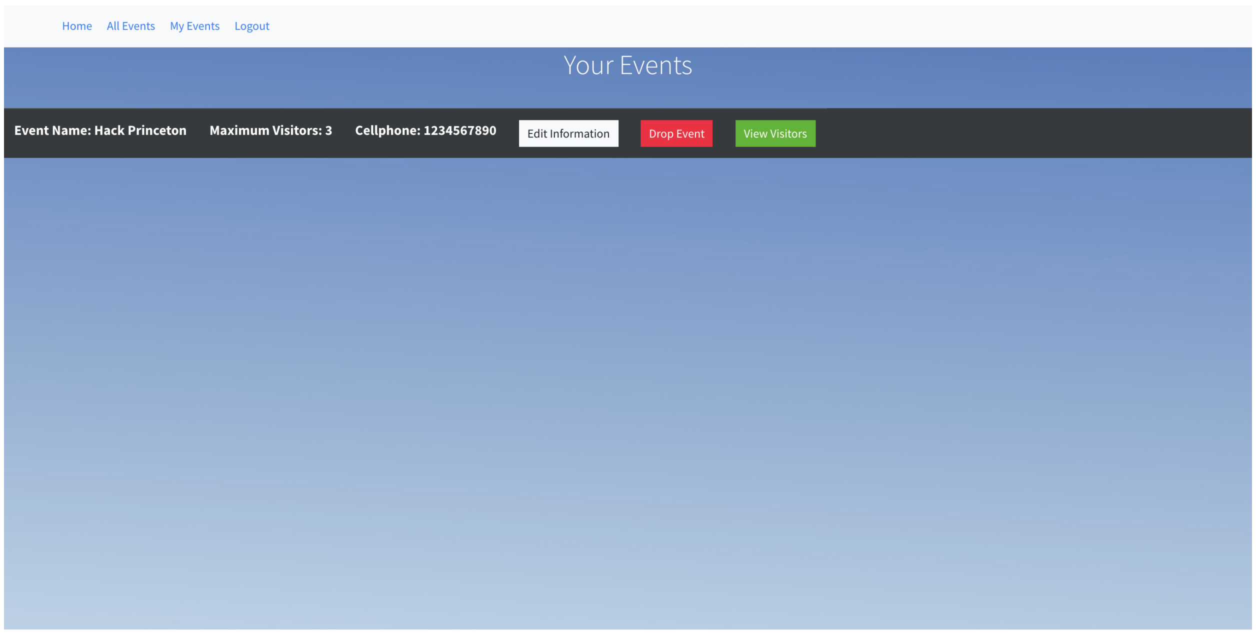

Host for Hack Princeton

First Name	Joe
Last Name	Schmo
Room Number (Ex: Little Hall 99)	Little Hall 99
Cellphone Number:	1234567890
Max Number of Visitors:	3
Gender	[✓ Male] Female
Opposite Gender Visitors	[✓ Yes] No

15) The user should be prompted with a "Host for Hack Princeton" modal. For the first name, the user will input "Joe," for the last name they will input "Schmo," "Little Hall 99" for the Room Number, "1234567890" for the cellphone number, 3 for max number of visitors, Male for gender, and Yes for opposite gender visitors. The user should then select "Sign Up."

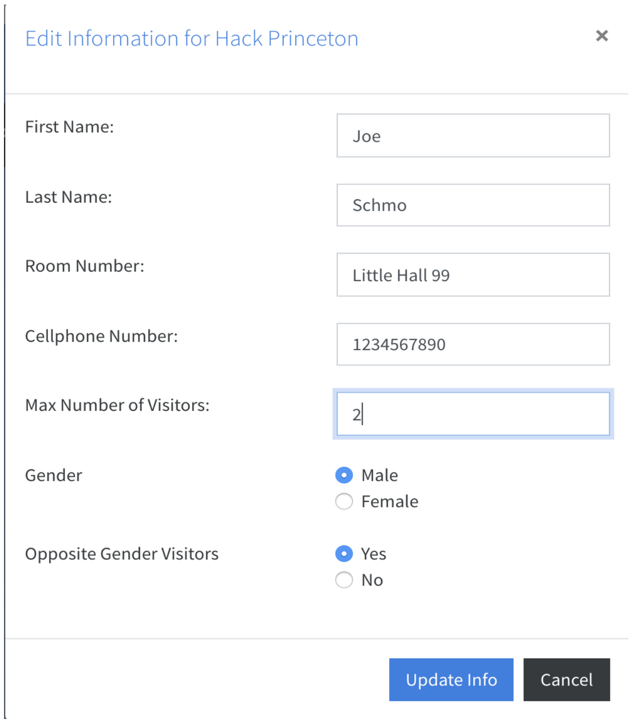


16) The user should be directed back to the "All events page," and Hack Princeton should no longer appear. The user should then select the "My Events" tab on the navigation bar.



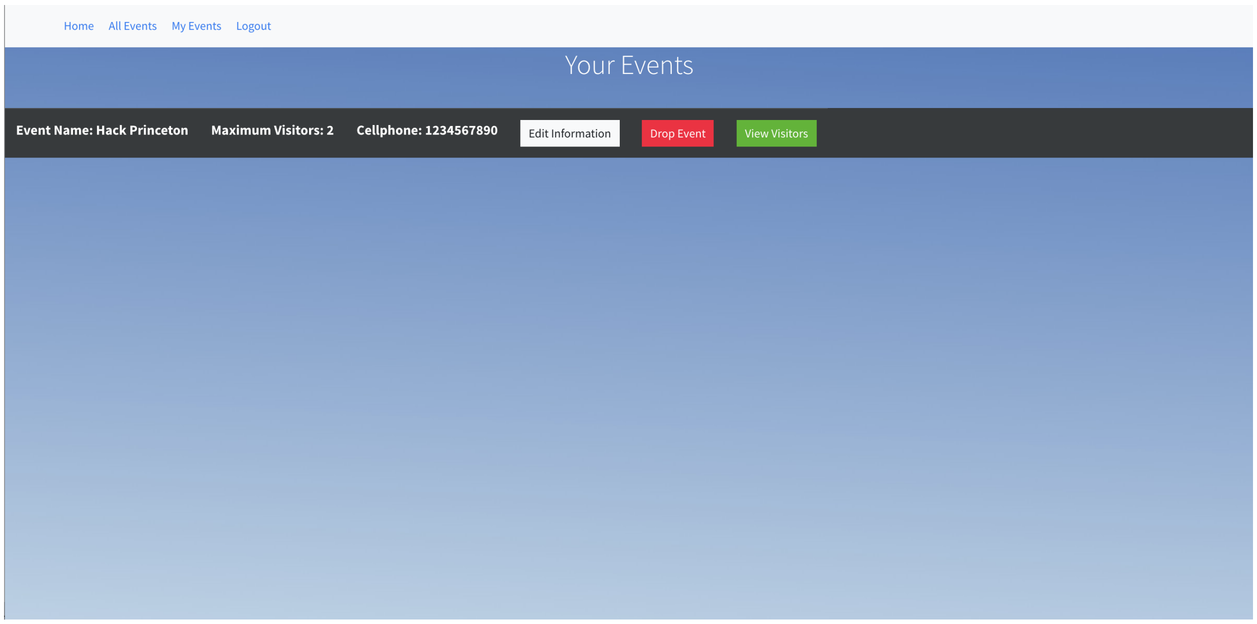
Event Name: Hack Princeton Maximum Visitors: 3 Cellphone: 1234567890

17) On the My Events page, the user should be able to a Table Row with "Hack Princeton" along with the other information regarding the hosting on the page. They should then click the "Edit information" button in the table row.



First Name:	Joe
Last Name:	Schmo
Room Number:	Little Hall 99
Cellphone Number:	1234567890
Max Number of Visitors:	2
Gender	[✓ Male] Female
Opposite Gender Visitors	[✓ Yes] No

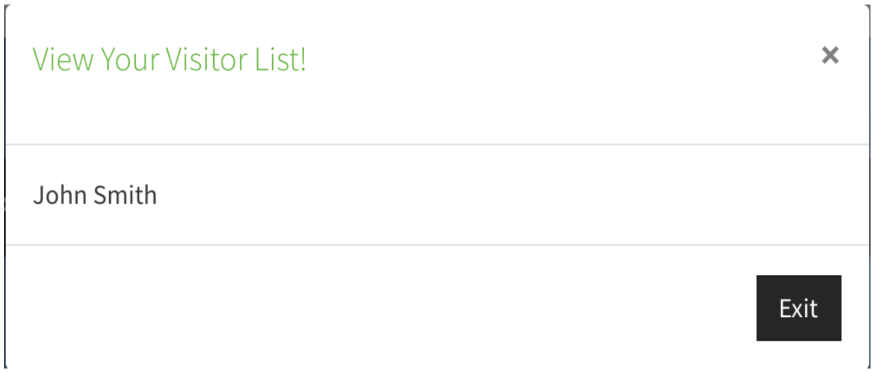
18) The user will be presented with an "Edit Information" Modal, where they should change the max number of visitors from 3 to 2, and then click update Info.



Event Name: Hack Princeton Maximum Visitors: 2 Cellphone: 1234567890

19) The user should be able to now see the updated information, with 2 maximum visitors, on the My Events page.

20) At this point in the process, the user will refer to the *TigerNest: Visitor Flow* document by Michelle Yuen, and should progress through all the stages outlined. When the user has completed this, they should return back to the Host my events page from step 19 in this tutorial, and select “View Visitors.”



John Smith

21) The user will now see “John Smith” under their visitor list.

Code:

Database.py

```
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from flask_marshmallow import Marshmallow
from flask_cors import CORS
from flask_bcrypt import Bcrypt
from flask_jwt_extended import (
    JWTManager, jwt_required, create_access_token,
    get_jwt_identity
)
import os
import hashlib
import json
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail
import time
import jwt

from flask_heroku import Heroku

basedir = os.path.abspath(os.path.dirname(__file__))

app = Flask(__name__)

# Setup the Flask-JWT-Extended extension
app.config['JWT_SECRET_KEY'] = 'super-secret'  # Change this!
app.config['JWT_ACCESS_TOKEN_EXPIRES'] = 60 * 60 * 24 # expire after 1 day
jwt_flask = JWTManager(app)

bcrypt = Bcrypt(app)
CORS(app)
#postgres://vhhabiabmrdtya:d2d626af16f4894a2a656de2ca8158f4414687b9b84ea08f801c2c07d774c6b8@ec2-54-243-2
41-62.compute-1.amazonaws.com:5432/d9d1mlh0cjs2lf

#app.config['SQLALCHEMY_DATABASE_URI'] =
"postgres://vhhabiabmrdtya:d2d626af16f4894a2a656de2ca8158f4414687b9b84ea08f801c2c07d774c6b8@ec2-54-243-2
41-62.compute-1.amazonaws.com:5432/d9d1mlh0cjs2lf"

#app.config['SQLALCHEMY_DATABASE_URI'] = "postgresql://localhost/tigernest"
app.config['SQLALCHEMY_DATABASE_URI'] = os.environ['DATABASE_URL']
#app.config['SQLALCHEMY_DATABASE_URI'] = os.environ['DATABASE_URL']

heroku = Heroku(app)
db = SQLAlchemy(app)
ma = Marshmallow(app)

@app.route("/getRegCode", methods = ["GET"])
def get_code():
        result = {"regCode": "2sdi319230d8208sd"}
        return jsonify(result)
```

```
class EventOrganizer(db.Model):
        event_organizer_id = db.Column(db.Integer, primary_key = True)
        netid = db.Column(db.Unicode, unique = True)
        firstname = db.Column(db.Unicode, unique = False)
        lastname = db.Column(db.Unicode, unique = False)
        email = db.Column(db.Unicode, unique = False)
        password = db.Column(db.Unicode, unique = False)
        campus_organizations = db.Column(db.JSON, unique = False)

        def __init__(self, netid, firstname, lastname, email, password, campus_organizations):
                self.netid = netid
                self.firstname = firstname
                self.lastname = lastname
                self.email = email
                self.password = password
                self.campus_organizations = campus_organizations


class EventOrganizerSchema(ma.Schema):
        class Meta:
                fields = ('event_organizer_id', 'netid', 'firstname', 'lastname', 'email', 'password',
'campus_organizations')

event_organizer_schema = EventOrganizerSchema()
event_organizers_schema = EventOrganizerSchema(many = True)

# Endpoint to add new event organizer
@app.route("/event_organizer", methods=["POST"])
def event_organizer_add():
        netid = request.json['netid']
        firstname = request.json['firstname']
        lastname = request.json['lastname']
        email = request.json['email']
        password = request.json['password']
        campus_organizations = request.json['campus_organizations']

        new_event_organizer = EventOrganizer(netid, firstname, lastname, email, password, campus_organizations)
        db.session.add(new_event_organizer)
        db.session.commit()
        return event_organizer_schema.jsonify(new_event_organizer)

# Endpoint to retrieve existing event organizer from the database
@app.route("/event_organizer/<event_organizer_id>", methods=["GET"])
def event_organizer_get(event_organizer_id):
        event_organizer = EventOrganizer.query.get(event_organizer_id)
        return event_organizer_schema.jsonify(event_organizer)

@app.route("/event_organizer/email/<email>", methods=["GET"])
def event_organizer_get_email(email):
        event_organizer = EventOrganizer.query.filter_by(email=email).first()
        return event_organizer_schema.jsonify(event_organizer)

@app.route("/event_organizer/netidVerify/<netid>", methods=["GET"])
def event_organizer_verify_netid(netid):
        event_organizer = EventOrganizer.query.filter_by(netid=netid).first()
        #return event_organizer_schema.jsonify(event_organizer)
        #return event_organizer_schema.jsonify(event_organizer)
```

```
	return event_organizer_schema.jsonify(event_organizer)


@app.route("/event_organizer/getCount", methods=["GET"])
def event_organizer_get_count():
	count = session.query(EventOrganizer.netid).count()
	return count
#------------------------------------------------------------------------------------------
class Event(db.Model):
	event_id = db.Column(db.Integer, primary_key = True)
	name = db.Column(db.Unicode, unique = False)
	start_date = db.Column(db.Unicode, unique = False)
	start_time = db.Column(db.Unicode, unique = False)
	end_date = db.Column(db.Unicode, unique = False)
	end_time = db.Column(db.Unicode, unique = False)
	description = db.Column(db.Unicode, unique = False)
	location = db.Column(db.Unicode, unique = False)
	expected_number_visitors = db.Column(db.Integer, unique = False)
	number_of_hosts = db.Column(db.Integer, unique = False)
	hosts = db.Column(db.JSON, unique = False)
	hosting_organization = db.Column(db.Unicode, unique=False)
	organizer_id = db.Column(db.Unicode, unique=False)
	organizer_netid = db.Column(db.Unicode, unique=False)
	event_stage = db.Column(db.Integer, unique=False)

	def __init__(self, name, start_date, end_date, location, expected_number_visitors, number_of_hosts, hosts,
start_time, end_time, description, hosting_organization, organizer_netid, organizer_id, event_stage):
		self.name = name
		self.start_date = start_date
		self.end_date = end_date
		self.location = location
		self.expected_number_visitors = expected_number_visitors
		self.number_of_hosts = number_of_hosts
		self.hosts = hosts
		self.start_time = start_time
		self.end_time = end_time
		self.description = description
		self.hosting_organization = hosting_organization
		self.organizer_id = organizer_id
		self.organizer_netid = organizer_netid
		self.event_stage = event_stage

class EventSchema(ma.Schema):
	class Meta:
		fields = ('event_id', 'name', 'start_date', 'end_date', 'location', 'expected_number_visitors',
'number_of_hosts', 'hosts', 'start_time', 'end_time', 'description', 'hosting_organization', 'organizer_netid', 'organizer_id',
'event_stage')

event_schema = EventSchema()
events_schema = EventSchema(many = True)

@app.route("/event", methods=["POST"])
def event_add():
	name = request.json['name']
	start_date = request.json['start_date']
	end_date = request.json['end_date']
	location = request.json['location']
```

```
		expected_number_visitors = request.json['expected_number_visitors']
		number_of_hosts = request.json['number_of_hosts']
		hosts = request.json['hosts']
		start_time = request.json['start_time']
		end_time = request.json['end_time']
		description = request.json['description']
		hosting_organization = request.json['hosting_organization']
		organizer_netid = request.json['organizer_netid']
		organizer_id = request.json['organizer_id']
		event_stage = 0

		new_event = Event(name, start_date, end_date, location, expected_number_visitors, number_of_hosts, hosts,
start_time, end_time, description, hosting_organization, organizer_netid, organizer_id, event_stage)
		db.session.add(new_event)
		db.session.commit()
		return event_schema.jsonify(new_event)

@app.route("/event/update/<event_id>", methods=["POST"])
def event_update_info(event_id):

		event = Event.query.get(event_id)
		event.name = request.json['name']
		event.start_date = request.json['start_date']
		event.end_date = request.json['end_date']
		event.location = request.json['location']
		event.expected_number_visitors = request.json['expected_number_visitors']
		event.number_of_hosts = request.json['number_of_hosts']
		event.hosts = request.json['hosts']
		event.start_time = request.json['start_time']
		event.end_time = request.json['end_time']
		event.description = request.json['description']
		event.hosting_organization = request.json['hosting_organization']
		#event.organizer_id = request.json['organizer_id']

		db.session.commit()
		return event_schema.jsonify(event)

@app.route("/event/stage/visitor_signup/<event_id>", methods=["POST"])
def event_update_stage_visitor_signup(event_id):
		event = Event.query.get(event_id)
		event.event_stage = 1;

		db.session.commit()
		return event_schema.jsonify(event)

@app.route("/event/stage/close_signup/<event_id>", methods=["POST"])
def event_update_stage_close_signup(event_id):
		event = Event.query.get(event_id)
		event.event_stage = 2;

		db.session.commit()
		return event_schema.jsonify(event)


@app.route("/event/<event_id>", methods=["GET"])
def event_get(event_id):
		event = Event.query.get(event_id)
		return event_schema.jsonify(event)

@app.route("/event/addHost/<event_id>", methods=["POST"])
def event_add_host(event_id):
		event = Event.query.get(event_id)
		event.number_of_hosts = event.number_of_hosts + 1
		db.session.commit()
		return event_schema.jsonify(event)

@app.route("/event/sort_date", methods=["GET"])
def event_sort_date():
		events = Event.query
		result = events.order_by(Event.start_date).all()
		return events_schema.jsonify(result)

@app.route("/event/most_recently_added", methods=["GET"])
def event_most_recent():
		events = Event.query
		result = events.order_by(Event.event_id.desc()).first()
		return events_schema.jsonify(result)

@app.route("/event/organizersEvents/<organizer_id>", methods=["GET"])
def event_get_organizers_events(organizer_id):
		result = Event.query.filter_by(organizer_id=organizer_id)
		return events_schema.jsonify(result)

@app.route("/event/organizersEvents/netid/<organizer_netid>", methods=["GET"])
def event_get_organizers_events_netid(organizer_netid):
		result = Event.query.filter_by(organizer_netid=organizer_netid).order_by(Event.start_date).all()
		return events_schema.jsonify(result)

@app.route("/event/delete/<event_id>", methods=["DELETE"])
def event_delete(event_id):
		event = Event.query.get(event_id)
		db.session.delete(event)
		db.session.commit()
		return event_schema.jsonify(event)


#------------------------------------------------------------------------------------------
class Host(db.Model):
		host_id = db.Column(db.Integer, primary_key = True)
		netid = db.Column(db.Unicode, unique = True)
		name = db.Column(db.Unicode, unique = False)
		email = db.Column(db.Unicode, unique = False)
		campus_organizations = db.Column(db.Unicode, unique = False)
		hosting_address = db.Column(db.Unicode, unique = False)
		max_visitors = db.Column(db.Integer, unique = False)
		gender = db.Column(db.Unicode, unique = False)
		same_gender = db.Column(db.Boolean, unique = False)
		expandable = db.Column(db.Boolean, unique = False)
		additional_visitors = db.Column(db.Integer, unique = False)

		def __init__(self, netid, name, email, campus_organizations, hosting_address, max_visitors, gender,
same_gender, expandable, additional_visitors):
				self.netid = netid
				self.name = name
		self.email = email
		self.campus_organizations = campus_organizations
		self.hosting_address = hosting_address
		self.max_visitors = max_visitors
		self.gender = gender
		self.same_gender = same_gender
		self.expandable = expandable
		self.additional_visitors = additional_visitors

class HostSchema(ma.Schema):
	class Meta:
		fields = ('host_id', 'netid', 'name', 'email', 'campus_organizations', 'hosting_address', 'max_visitors',
'gender', 'same_gender', 'expandable', 'additional_visitors')

host_schema = HostSchema()
hosts_schema = HostSchema(many = True)

@app.route("/host", methods=["POST"])
def host_add():
	netid = request.json['netid']
	name = request.json['name']
	email = request.json['email']
	campus_organizations = request.json['campus_organizations']
	hosting_address = request.json['hosting_address']
	max_visitors = request.json['max_visitors']
	gender = request.json['gender']
	same_gender = request.json['same_gender']
	expandable = request.json['expandable']
	additional_visitors = request.json['additional_visitors']

	new_host = Host(netid, name, email, campus_organizations, hosting_address, max_visitors, gender,
same_gender, expandable, additional_visitors)

	db.session.add(new_host)
	db.session.commit()
	return host_schema.jsonify(new_host)

@app.route("/host/<host_id>", methods=["GET"])
def host_get(host_id):
	host = Host.query.get(host_id)
	return host_schema.jsonify(host)

#----------------------------------------------------------------------------------------------------------------------------------------
"""class VisitorPairing(db.Model):
	visitorpairing_id = db.Column(db.Integer, primary_key=True)
	visitor_id = db.Column(db.Integer, unique=False)
	event_id = db.Column(db.Integer, unique=False)
	host_id = db.Column(db.Integer, unique=False)

class VisitorPairingSchema(db.Model):
	class Meta:
		fields = ('visitorpairing_id', 'visitor_id', 'event_id', 'host_id')

visitorpairing_schema = VisitorPairingSchema()
visitorpairings_schema = VisitorPairingSchema(many = True)"""
#----------------------------------------------------------------------------------------------------------------------------------------
class Pairing(db.Model):
	pairing_id = db.Column(db.Integer, primary_key=True)
	event_id = db.Column(db.Integer, unique=False)
	host_gender = db.Column(db.Unicode, unique=False)
	same_gender_room = db.Column(db.Boolean, unique=False)
	host_room_num = db.Column(db.Unicode, unique=False)
	max_visitors = db.Column(db.Integer, unique=False)
	num_visitors = db.Column(db.Integer, unique =False)
	host_first_name = db.Column(db.Unicode, unique=False)
	host_last_name = db.Column(db.Unicode, unique=False)
	host_cellphone = db.Column(db.Unicode, unique=False)
	host_netid = db.Column(db.Unicode, unique=False)
	event_name = db.Column(db.Unicode, unique=False)

	def __init__(self, event_id, host_gender, same_gender_room, host_room_num, max_visitors, num_visitors,
host_first_name, host_last_name, host_cellphone, host_netid, event_name):
		self.event_id = event_id
		self.host_gender = host_gender
		self.same_gender_room = same_gender_room
		self.host_room_num = host_room_num
		self.max_visitors = max_visitors
		self.num_visitors = num_visitors
		self.host_first_name = host_first_name
		self.host_last_name = host_last_name
		self.host_cellphone = host_cellphone
		self.host_netid = host_netid
		self.event_name = event_name


class PairingSchema(ma.Schema):
	class Meta:
		fields = ('pairing_id', 'event_id', 'host_gender', 'same_gender_room', 'host_room_num', 'max_visitors',
'num_visitors', 'host_first_name', 'host_last_name', 'host_cellphone', 'host_netid', 'event_name')

pairing_schema = PairingSchema()
pairings_schema = PairingSchema(many = True)

@app.route("/pairing", methods=["POST"])
def pairing_add():
	event_id = request.json['event_id']
	host_gender = request.json['host_gender']
	same_gender_room = request.json['same_gender_room']
	host_room_num = request.json['host_room_num']
	max_visitors = int(request.json['max_visitors'])
	num_visitors = 0
	host_first_name = request.json['host_first_name']
	host_last_name = request.json['host_last_name']
	host_cellphone = request.json['host_cellphone']
	host_netid = request.json['host_netid']
	event_name = request.json['event_name']


	new_pairing = Pairing(event_id, host_gender, same_gender_room, host_room_num, max_visitors,
num_visitors, host_first_name, host_last_name, host_cellphone, host_netid, event_name)

	db.session.add(new_pairing)
	db.session.commit()
	return pairing_schema.jsonify(new_pairing)
```

```
@app.route("/pairing/<pairing_id>", methods=["GET"])
def pairing_get(pairing_id):
        pairing = Pairing.query.get(pairing_id)
        return pairing_schema.jsonify(pairing)


@app.route("/pairing/events_for_host/<host_netid>", methods=["GET"])
def pairing_get_event_for_host(host_netid):
        pairings = Pairing.query.filter_by(host_netid=host_netid).all()
        return pairings_schema.jsonify(pairings)

@app.route("/pairing/hosts_for_event/<event_id>", methods=["GET"])
@jwt_required
def pairing_get_host_for_event(event_id):
        pairings = Pairing.query.filter_by(event_id=event_id).all()
        return pairings_schema.jsonify(pairings)

@app.route("/pairing/addVisitor/<pairing_id>", methods=["POST"])
@jwt_required
def pairing_add_visitor(pairing_id):
        pairing = Pairing.query.get(pairing_id)
        pairing.num_visitors = pairing.num_visitors + 1
        db.session.commit()
        return pairing_schema.jsonify(pairing)

@app.route("/pairing/removeVisitor/<pairing_id>", methods=["POST"])
@jwt_required
def pairing_remove_visitor(pairing_id):
        pairing = Pairing.query.get(pairing_id)
        pairing.num_visitors = pairing.num_visitors - 1
        db.session.commit()
        return pairing_schema.jsonify(pairing)

@app.route("/pairing/delete/<pairing_id>", methods=["DELETE"])
def pairing_delete(pairing_id):
        pairing = Pairing.query.get(pairing_id)
        db.session.delete(pairing)
        db.session.commit()
        return pairing_schema.jsonify(pairing)

@app.route("/pairing/update/<pairing_id>", methods=["POST"])
def pairing_update(pairing_id):
        pairing = Pairing.query.get(pairing_id)
        pairing.host_gender = request.json['host_gender']
        pairing.same_gender_room = request.json['same_gender_room']
        pairing.host_room_num = request.json['host_room_num']
        pairing.max_visitors = request.json['max_visitors']
        pairing.host_first_name = request.json['host_first_name']
        pairing.host_last_name = request.json['host_last_name']
        pairing.host_cellphone = request.json['host_cellphone']

        db.session.commit()
        return pairing_schema.jsonify(pairing)

#------------------------------------------------------------------------------------------------------------------------------------------
class VisitorPairing(db.Model):
	visitor_pairing_id = db.Column(db.Integer, primary_key = True)
	visitor_id = db.Column(db.Integer, unique=False)
	visitor_email = db.Column(db.Unicode, unique=False)
	event_id = db.Column(db.Integer, unique=False)
	pairing_id = db.Column(db.Integer, unique=False)


	def __init__(self, visitor_id, visitor_email, event_id, pairing_id):
		self.visitor_id = visitor_id
		self.visitor_email = visitor_email
		self.event_id = event_id
		self.pairing_id = pairing_id

class VisitorPairingSchema(ma.Schema):
	class Meta:
		fields=('visitor_pairing_id', 'visitor_id', 'visitor_email', 'event_id', 'pairing_id')

visitor_pairing_schema = VisitorPairingSchema()
visitor_pairings_schema = VisitorPairingSchema(many = True)


@app.route("/visitor_pairing", methods=["POST"])
@jwt_required
def visitor_pairing_add():
	visitor_id = request.json['visitor_id']
	visitor_email = request.json['visitor_email']
	event_id = request.json['event_id']
	pairing_id = int(request.json['pairing_id'])
	new_visitor_pairing = VisitorPairing(visitor_id, visitor_email, event_id, pairing_id)
	db.session.add(new_visitor_pairing)
	db.session.commit()
	return visitor_pairing_schema.jsonify(new_visitor_pairing)

@app.route("/visitor_pairing/delete/<visitor_pairing_id>", methods=["DELETE"])
def visitor_pairing_delete(visitor_pairing_id):
	visitor_pairing = VisitorPairing.query.get(visitor_pairing_id)
	db.session.delete(visitor_pairing)
	db.session.commit()
	return visitor_pairing_schema.jsonify(visitor_pairing)

@app.route("/visitor_pairing/guests_in_room/<pairing_id>", methods=["GET"])
@jwt_required
def visitor_pairing_get_guests_in_room(pairing_id):
	count = VisitorPairing.query.filter_by(pairing_id=pairing_id).count()
	return str(count)

@app.route("/visitor_pairing/pairings_for_event/<event_id>", methods=["GET"])
def visitor_pairing_get_for_events(event_id):
	visitor_pairings = VisitorPairing.query.filter_by(event_id=event_id).all()
	return visitor_pairings_schema.jsonify(visitor_pairings)

@app.route("/visitor_pairing/all_hosts/<pairing_id>", methods=["GET"])
def visitor_pairing_get_hosts(pairing_id):
	visitor_pairings = VisitorPairing.query.filter_by(pairing_id=pairing_id).all()
	return visitor_pairings_schema.jsonify(visitor_pairings)

@app.route("/visitor_pairing/filter_eligibilities/<event_id>/<visitor_id>", methods=["GET"])
```

```
def visitor_pairing_filter_eligibilities(event_id, visitor_id):
        visitor_pairing = VisitorPairing.query.filter_by(event_id=event_id, visitor_id=visitor_id).first()
        return visitor_pairing_schema.jsonify(visitor_pairing)


#------------------------------------------------------------------------------------------------------------------------------------------
class Eligibilities(db.Model):
        eligibility_id = db.Column(db.Integer, primary_key = True)
        visitor_email = db.Column(db.Unicode, unique=False)
        event_id = db.Column(db.Unicode, unique=False)
        event_name = db.Column(db.Unicode, unique=False)
        signed_up = db.Column(db.Boolean, unique=False)

        def __init__(self, visitor_email, event_id, event_name, signed_up):
                self.visitor_email = visitor_email
                self.event_id = event_id
                self.event_name = event_name
                self.signed_up = signed_up

class EligibilitySchema(ma.Schema):
        class Meta:
                fields=('eligibility_id', 'visitor_email', 'event_id', 'event_name', 'signed_up')


eligibility_schema = EligibilitySchema()
eligibilities_schema = EligibilitySchema(many = True)

@app.route("/eligibility", methods=["POST"])
def eligibility_add():
        visitor_email = request.json['visitor_email']
        event_id = request.json['event_id']
        event_name = request.json['event_name']
        signed_up = False
        new_eligibility = Eligibilities(visitor_email, event_id, event_name, signed_up)
        db.session.add(new_eligibility)
        db.session.commit()
        return eligibility_schema.jsonify(new_eligibility)

@app.route("/eligibility/<eligibility_id>", methods=["GET"])
def eligibility_get(eligibility_id):
        eligibility = Eligibilities.query.get(eligibility_id)
        return eligibility_schema.jsonify(eligibility)

@app.route("/eligibility/visitor_signup/<eligibility_id>", methods=["POST"])
def eligibility_visitor_signup(eligibility_id):
        eligibility = Eligibilities.query.get(eligibility_id)
        eligibility.signed_up = True
        db.session.commit()
        return eligibility_schema.jsonify(eligibility)

@app.route("/eligibility/visitor_signup_not/<eligibility_id>", methods=["POST"])
def eligibility_visitor_not_signup(eligibility_id):
        eligibility = Eligibilities.query.get(eligibility_id)
        eligibility.signed_up = False
        db.session.commit()
        return eligibility_schema.jsonify(eligibility)
```

```
@app.route("/eligibility/events_for_visitor/<visitor_email>", methods=["GET"])
def eligibility_events_for_visitor(visitor_email):
        eligibilities = Eligibilities.query.filter_by(visitor_email=visitor_email).all()
        return eligibilities_schema.jsonify(eligibilities)


#--------------------------------------------------------------------------------------------------------------------------------
class Visitor(db.Model):
        id = db.Column(db.Integer, primary_key = True)
        gender = db.Column(db.Unicode, unique = False)
        name = db.Column(db.Unicode, unique = False)
        same_gender = db.Column(db.Boolean, unique = False)
        university = db.Column(db.Unicode, unique = False)
        email = db.Column(db.Unicode, unique = True)
        password = db.Column(db.Unicode, unique = False)

        def __init__(self, gender, name, same_gender, university, email, password):
                self.gender = gender
                self.name = name
                self.same_gender = same_gender
                self.university = university
                self.email = email
                self.password = bcrypt.generate_password_hash(password, 10).decode('utf8')

class VisitorSchema(ma.Schema):
        class Meta:
                fields = ('id', 'gender', 'name', 'same_gender', 'university', 'email', 'password')

visitor_schema = VisitorSchema()
visitors_schema = VisitorSchema(many = True)

# def authenticate(username, password):
#       visitor = Visitor.query.filter_by(email=username).first()
#       if visitor and bcrypt.check_password_hash(visitor.password, password):
#               return visitor

# def identity(payload):
#       id = payload['identity']
#       return Visitor.query.filter_by(id=id).first()

@app.route("/visitor", methods=["POST"])
def visitor_add():
        gender = request.json['gender']
        name = request.json['name']
        gender_same = request.json['same_gender']

        same_gender = False
        if (gender_same == "True"):
                same_gender = True
        university = request.json['university']
        email = request.json['email']
        password = request.json['password']

        new_visitor = Visitor(gender, name, same_gender, university, email, password)
        db.session.add(new_visitor)
        db.session.commit()
        identity = {
            "id": new_visitor.id,
            "email": new_visitor.email
        }
        access_token = create_access_token(identity=identity)
        return jsonify(access_token=access_token), 200

@app.route("/visitor/login", methods=["POST"])
def visitor_login():
        email = request.json['email']
        password = request.json['password']
        visitor = Visitor.query.filter_by(email=email).first()

        if visitor and bcrypt.check_password_hash(visitor.password, password):
            identity = {
                "id": visitor.id,
                "email": visitor.email
            }
            access_token = create_access_token(identity=identity)
            return jsonify(access_token=access_token), 200

        return jsonify({"msg": "Bad username or password"}), 401

@app.route("/visitor/reset", methods=["POST"])
def visitor_reset():
        email = request.json['email']
        visitor = Visitor.query.filter_by(email=email).first()

        if visitor:
            # Expire token in 60 minutes
            reset_token = jwt.encode({"id": visitor.id, "exp": int(time.time()) + 60*60},
app.config['JWT_SECRET_KEY'], algorithm='HS256')

            message = Mail(
                from_email='from_email@example.com',
                to_emails=email,
                subject='Password Reset',

html_content='https://tiger-nest.herokuapp.com/visitor/reset?resetToken='+reset_token.decode("utf-8") )
            try:
                sg = SendGridAPIClient(os.environ.get('SENDGRID_API_KEY'))
                response = sg.send(message)
                print(response.status_code)
                print(response.body)
                print(response.headers)
                return jsonify(), 200
            except Exception as e:
                print(e.message)
                return jsonify(), 500

        return jsonify({"msg": "Invalid user"}), 401


@app.route("/visitor/change-password", methods=["POST"])
def visitor_change_password():
        password = request.json['password']
        reset_token = jwt.decode(request.json['resetToken'], app.config['JWT_SECRET_KEY'], algorithms=['HS256'])
        visitor = Visitor.query.get(reset_token["id"])
        print(visitor)

        if visitor:
                visitor.password = bcrypt.generate_password_hash(password, 10).decode('utf8')
                db.session.commit()
                return jsonify({"msg": "Updated"}), 200

        return jsonify({"msg": "Invalid user"}), 401


@app.route("/visitor/<visitor_id>", methods=["GET"])
def visitor_get(visitor_id):
        visitor = Visitor.query.get(visitor_id)
        return visitor_schema.jsonify(visitor)

@app.route('/visitor/data')
@jwt_required
def protected():
        visitor_id = get_jwt_identity()['id']
        visitor = Visitor.query.get(visitor_id)
        return visitor_schema.jsonify(visitor)


db.create_all()
#----------------------------------------------------------------------------------------------------
if __name__ == '__main__':
        app.run(debug=True, host='0.0.0.0', port=os.environ.get("PORT", 5000))
```

Server.js

```
const express = require('express');
const next = require('next');
let session = require('cookie-session')

const dev = process.env.NODE_ENV !== 'production';
const app = next({ dev });
const handle = app.getRequestHandler();
var CentralAuthenticationService = require('cas');


app.prepare()
  .then(() => {
    const server = express();
    var casURL = 'https://fed.princeton.edu/cas/'
    var cas = new CentralAuthenticationService({
      base_url: casURL,
      service: "http://tiger-nest2.herokuapp.com" + "/verify"
    })
    server.use(session({
      secret: 'abcdefghijklmnop',
      maxAge: 24 * 60 * 60 * 1000 * 365,
      cookie: { secure: false }
    }))
    server.set('json spaces', 2);

    server.get('/login', function (req, res) {
      // Save the user's redirection destination to a cookie
      if (typeof (req.query.redirect) === 'string') {
            req.session.redirect = req.query.redirect
          }
      // Redirect the user to the CAS server
      res.redirect(casURL + 'login?service=' + "http://tiger-nest2.herokuapp.com/verify")

    })
    server.get('/logout', function (req, res) {
      req.session = null
      res.cookie('netid', null)
      res.redirect(casURL + 'logout?url=http://tiger-nest.herokuapp.com')
    })
    server.get('/verify', function(req, res) {
      // Check if the user has a redirection destination
      let redirectDestination = req.session.redirect || '/'
      // If the user already has a valid CAS session then send them to their destination
      if (req.session.cas) {
        res.redirect(redirectDestination)
        return
      }
      var ticket = req.query.ticket

      // If the user does not have a ticket then send them to the homepage
      if (typeof (ticket) === 'undefined') {
        res.redirect('/')
        return
      }
      // Check if the user's ticket is valid
      cas.validate(ticket, function (err, status, netid) {
        if (err) {
          console.log(err)
          res.sendStatus(500)
          return
        }
          req.session.cas = {
                status: status,
                netid: netid
          };
          res.cookie('netid', netid)
          //req.session.cookie.netid = netid;
          res.redirect(redirectDestination);
     }) });
    server.get('/eventOrganizerRegister', function(req, res){
      return handle(req, res);
    });
    server.get('/test', function (req, res) {
      res.send({yo: req.session.cas.netid});
    });

    server.get('/netid', function (req, res) {
      // Save the user's redirection destination to a cookie
      if (req.session.cas) {
        res.send({netid: req.session.cas.netid});
        //console.log("yeeeeeeeeeeeet");
      }
      else res.redirect("/login?redirect=/")
    })
    server.get('/', function (req, res) {
        res.redirect("https://tiger-nest.herokuapp.com")
    })

    server.get('*/bootstrap.css',
        function (req, res) {
            return handle(req, res);
        }
    )
    server.get('/static/background.jpg',
        function (req, res) {
            return handle(req, res);
        }
    )
    server.get('*', function (req, res) {
        if (req.session.cas){
            return handle(req, res);
        }
        res.redirect("/login?redirect=/")
    })

    server.set('views', '/views');
    server.set('view engine', 'js');
    server.engine('js', require('express-react-views').createEngine());


    server.listen(process.env.PORT || 3000, (err) => {
        if (err) {
            throw err;
        }
        //console.log('> Ready on http://localhost:3000');
    });

})
.catch((ex) => {
    console.error(ex.stack);
    process.exit(1);
});
```

My Events.js

```
import './bootstrap.css';
import React from 'react'

import Link from 'next/link'
import Head from '../components/head'
import NavBar from '../components/nav'
import fetch from 'isomorphic-unfetch'
import { Button, Modal, ModalHeader, ModalBody, ModalFooter, Row, Col, Input, Form, FormText, CustomInput } from
'reactstrap';
import { Nav, NavItem, NavLink, TabContent, TabPane, Card, CardImg, CardText, CardHeader, CardBody, CardTitle,
CardSubtitle, CardDeck} from 'reactstrap';
import ReactFileReader from 'react-file-reader';
import EventOrganizerRegister from './eventOrganizerRegister'

import Router from 'next/router'
import Cookies from 'js-cookie';

//const database_url = "http://localhost:5000"
//const server_url = "http://localhost:3000"

const server_url = "https://tiger-nest2.herokuapp.com"

const database_url = "https://tigernest-backend.herokuapp.com"

const axios = require('axios')

var divStyle = {
  color: 'white'
  //color: 'dodgerblue'
};

var divStyle2 = {
  color: 'black'
  //color: 'dodgerblue'
};

var divStyle3 = {
  //color: 'black'
  color: 'dodgerblue'
};

var divStyle4 = {
  //color: 'black'
  color: 'seagreen'
};


class eventList extends React.Component {
  constructor(props, context){
    super(props, context);
    this.state = {
      modal: false,
      editModal: false,
      deleteModal: false,
      visitorEmails: [],
      event_org_id: -1,
      user_verified: false,
      visitorSignupModal: false,
      finalizePairingsModal: false,
      activeTab: '1',
      viewDetailsModal: false,
      pairings_for_event:[],
      current_event: {
        name:"",
        start_time:"",
        end_time:"",
        description:"",
        hosting_organization:"",
        expected_number_visitors:"",
        event_id:""
       }
    };
    this.addOrganizer = this.addOrganizer.bind(this);
    this.toggle = this.toggle.bind(this);
    this.addEvent = this.addEvent.bind(this);
    this.editEvent = this.editEvent.bind(this);
    this.deleteEvent = this.deleteEvent.bind(this);
    this.handleFiles = this.handleFiles.bind(this);
    this.editModalToggle = this.editModalToggle.bind(this);
    this.deleteModalToggle = this.deleteModalToggle.bind(this);
    this.toggleTab = this.toggleTab.bind(this);
    this.beginVisitorSignup = this.beginVisitorSignup.bind(this);
    this.visitorSignupToggle = this.visitorSignupToggle.bind(this);
    this.finalizePairings = this.finalizePairings.bind(this);
    this.finalizePairingsToggle = this.finalizePairingsToggle.bind(this);
    this.viewDetailsToggle = this.viewDetailsToggle.bind(this);
    this.refreshPage = this.refreshPage.bind(this);
    //this.verifyUser = this.verifyUser.bind(this);
  }
  toggleTab(tab){
    if (this.state.activeTab !== tab) {
      this.setState({
        activeTab: tab
      });
    }
  }
  refreshPage(){
    window.location.reload();
```

```
}
async beginVisitorSignup(){
  const res = await fetch(database_url + '/event/stage/visitor_signup/' + this.state.current_event.event_id, {
      method: "POST",
      headers: {
          'Accept': 'application/json',
          "Content-Type": "application/json"
      }
  });
  this.visitorSignupToggle();
  this.refreshPage();

}
async finalizePairings(){
  const res = await fetch(database_url + '/event/stage/close_signup/' + this.state.current_event.event_id, {
      method: "POST",
      headers: {
          'Accept': 'application/json',
          "Content-Type": "application/json"
      }
  });
  this.finalizePairingsToggle();
  this.refreshPage();

}
async viewDetailsToggle(){
  if (!this.state.viewDetailsModal)
  {
    let val = event.target.value;

    const res = await fetch(database_url + '/event/' + val, {
         method: "GET",
         headers: {
             "Content-Type": "text/plain",
             "Access-Control-Allow-Origin": "*"
    }})
    var data = await res.json()
    data = JSON.stringify(data)
    data = JSON.parse(data)
    this.setState(state => ({ current_event: data}));
    const res2 = await fetch(database_url + "/visitor_pairing/pairings_for_event/" + val, {
      method: "GET",
         headers: {
             "Content-Type": "text/plain",
             "Access-Control-Allow-Origin": "*"

    }})

    data = await res2.json()
    let pairingList = []



    for (let i = 0; i < data.length; i++)
    {
      let visitor_id = data[i]['visitor_id']
      let pairing_id = data[i]['pairing_id']

      let resVisitor = await fetch(database_url + '/visitor/' + visitor_id, {
         method: "GET",
         headers: {
             "Content-Type": "text/plain",
             "Access-Control-Allow-Origin": "*"
      }})
      var dataVisitor = await resVisitor.json()
      dataVisitor = JSON.stringify(dataVisitor)
```

```
        dataVisitor = JSON.parse(dataVisitor)

        let resHost = await fetch(database_url + '/pairing/' + pairing_id, {
          method: "GET",
          headers: {
              "Content-Type": "text/plain",
              "Access-Control-Allow-Origin": "*"
        }})
        var dataHost = await resHost.json()
        dataHost = JSON.stringify(dataHost)
        dataHost = JSON.parse(dataHost)

        let pairing_info = String("Host: " + dataHost['host_first_name'] + " " + dataHost['host_last_name'] + ", Visitor: " +
dataVisitor['name'] + "\n")
        pairingList.push(pairing_info)
      }

      //data = JSON.stringify(data);
      //data = JSON.parse(data);
      this.setState(state => ({ pairings_for_event: pairingList}));
    }

    this.setState(prevState => ({
      viewDetailsModal: !prevState.viewDetailsModal
    }));

  }
  async addOrganizer(){
  //console.log(document.forms["registerForm"]["netid"].value);

    let emailInput = document.forms["registerForm"]["email"].value;
    let passwordInput1 = document.forms["registerForm"]["password1"].value;
    let passwordInput2 = document.forms["registerForm"]["password2"].value;
    let firstnameInput = document.forms["registerForm"]["firstname"].value;
    let lastnameInput = document.forms["registerForm"]["lastname"].value;
    let registrationCode = document.forms["registerForm"]["regcode"].value;
    let netid = Cookies.get('netid');

    //
    const res = await fetch(database_url + "/getRegCode", {
        method: "GET",
        headers: {
            "Content-Type": "text/plain",
            "Access-Control-Allow-Origin": "*"
        }})

    var data = await res.json();
    data = JSON.stringify(data);

    data = JSON.parse(data);
    let trueRegCode = data['regCode'];

    /*if (passwordInput1 !== passwordInput2)
    {
        this.setState(state => ({ mismatchPassword: true}));
    }*/
    if (registrationCode !== trueRegCode)
    {
      this.setState(state => ({ wrongRegCode: true}));
    }
    else
    {
       let organizer_info = {
        "firstname": firstnameInput,
        "lastname": lastnameInput,
        "password": passwordInput1,
        "campus_organizations": "",
        "netid": netid,
        "email": emailInput,
       };
       const res = await fetch(database_url + '/event_organizer', {
        method: "POST",
        headers: {
            'Accept': 'application/json',
            "Content-Type": "application/json"
        },
        body: JSON.stringify(organizer_info)
      });
       Router.push("/eventOrganizerLogin");
    }
    this.refreshPage();
  }
  async deleteEvent(){
    const res = await fetch(database_url + '/event/delete/' + this.state.current_event.event_id, {
        method: "DELETE",
        headers: {
            'Accept': 'application/json',
            "Content-Type": "application/json"
        }
    });
    this.deleteModalToggle();
    this.refreshPage();

  }
  async editEvent(){
    //console.log(this.state.visitorEmails)
    //console.log(document.cookie)
     if (document.forms["eventEditForm"]["eventname"].value === "" || document.forms["eventEditForm"]["starttime"].value
=== "" || document.forms["eventEditForm"]["startdate"].value === "" || document.forms["eventEditForm"]["endtime"].value
=== "" || document.forms["eventEditForm"]["enddate"].value === "" ||
document.forms["eventEditForm"]["description"].value === "" || document.forms["eventEditForm"]["location"].value === "" ||
document.forms["eventEditForm"]["expectednum"].value === "" || document.forms["eventEditForm"]["hostingorg"].value
=== "")
    {
      alert("All fields are required!");
      return;
    }
    if (isNaN(parseInt(document.forms["eventEditForm"]["expectednum"].value)))
    {
      alert("Expected number of visitors must be an integer!");
      return;
    }

    let eventInfo = {
      "name": document.forms["eventEditForm"]["eventname"].value,
      "start_time": document.forms["eventEditForm"]["starttime"].value,
      "start_date": document.forms["eventEditForm"]["startdate"].value,
      "end_time": document.forms["eventEditForm"]["endtime"].value,
      "end_date": document.forms["eventEditForm"]["enddate"].value,
      "description": document.forms["eventEditForm"]["description"].value,
      "location": document.forms["eventEditForm"]["location"].value,
      "expected_number_visitors": parseInt(document.forms["eventEditForm"]["expectednum"].value),
      "number_of_hosts": 0,
      "hosts": "",
      "hosting_organization": document.forms["eventEditForm"]["hostingorg"].value,
      "organizer_id": 1
    };

    const res = await fetch(database_url + '/event/update/' + this.state.current_event.event_id, {
        method: "POST",
        headers: {
            'Accept': 'application/json',
            "Content-Type": "application/json"
        },
        body: JSON.stringify(eventInfo)
    });

    this.editModalToggle();
    this.refreshPage();


  }
  async finalizePairingsToggle(event){
    if (!this.state.finalizePairingsModal)
    {
      let val = event.target.value;

      const res = await fetch(database_url + '/event/' + val, {
          method: "GET",
          headers: {
            "Content-Type": "text/plain",
            "Access-Control-Allow-Origin": "*"
      }})
      var data = await res.json()
      data = JSON.stringify(data)
      data = JSON.parse(data)
      this.setState(state => ({ current_event: data}));
    }

      //console.log(data);


    this.setState(prevState => ({
      finalizePairingsModal: !prevState.finalizePairingsModal
    }));

  }
  async visitorSignupToggle(event){
    if (!this.state.visitorSignupModal)
    {
      let val = event.target.value;

      const res = await fetch(database_url + '/event/' + val, {
          method: "GET",
          headers: {
            "Content-Type": "text/plain",
            "Access-Control-Allow-Origin": "*"
      }})
      var data = await res.json()
      data = JSON.stringify(data)
      data = JSON.parse(data)
      this.setState(state => ({ current_event: data}));
    }

      //console.log(data);


    this.setState(prevState => ({
      visitorSignupModal: !prevState.visitorSignupModal
    }));

  }
  async deleteModalToggle(event){
    if (!this.state.deleteModal)
    {
      let val = event.target.value;

      const res = await fetch(database_url + '/event/' + val, {
          method: "GET",
          headers: {
            "Content-Type": "text/plain",
            "Access-Control-Allow-Origin": "*"
      }})
      var data = await res.json()
      data = JSON.stringify(data)
      data = JSON.parse(data)
      this.setState(state => ({ current_event: data}));
    }

      //console.log(data);


    this.setState(prevState => ({
      deleteModal: !prevState.deleteModal
    }));

  }
  async editModalToggle(event) {

    this.setState(state => ({ visitorEmails: ""}));
    //this.setState(state => ({ current_event: event.target.value}));
    if (!this.state.editModal)
    {
      let val = event.target.value;

      const res = await fetch(database_url + '/event/' + val, {
          method: "GET",
          headers: {
            "Content-Type": "text/plain",
            "Access-Control-Allow-Origin": "*"
      }})
      var data = await res.json()
      data = JSON.stringify(data)
      data = JSON.parse(data)
      this.setState(state => ({ current_event: data}));
    }


    this.setState(prevState => ({
      editModal: !prevState.editModal
    }));


  }
  async addEvent(){

    if (document.forms["eventCreateForm"]["eventname"].value === "" ||
document.forms["eventCreateForm"]["starttime"].value === "" || document.forms["eventCreateForm"]["startdate"].value
=== "" || document.forms["eventCreateForm"]["endtime"].value === "" ||
document.forms["eventCreateForm"]["enddate"].value === "" || document.forms["eventCreateForm"]["description"].value
=== "" || document.forms["eventCreateForm"]["location"].value === "" ||
document.forms["eventCreateForm"]["expectednum"].value === "" ||
document.forms["eventCreateForm"]["hostingorg"].value === "")
    {
      alert("All fields are required!");
      return;
    }
    if (isNaN(parseInt(document.forms["eventCreateForm"]["expectednum"].value)))
    {
      alert("Expected number of visitors must be an integer!");
      return;
    }

    const res1 = await fetch(database_url + '/event_organizer/netidVerify/' + Cookies.get('netid'), {
            method: "GET",
            headers: {
                "Content-Type": "text/plain",
                "Access-Control-Allow-Origin": "*"
            }})
      var data = await res1.json()

    var organizer_id = data['event_organizer_id']

    let eventInfo = {
      "name": document.forms["eventCreateForm"]["eventname"].value,
      "start_time": document.forms["eventCreateForm"]["starttime"].value,
      "start_date": document.forms["eventCreateForm"]["startdate"].value,
      "end_time": document.forms["eventCreateForm"]["endtime"].value,
      "end_date": document.forms["eventCreateForm"]["enddate"].value,
      "description": document.forms["eventCreateForm"]["description"].value,
      "location": document.forms["eventCreateForm"]["location"].value,
      "expected_number_visitors": parseInt(document.forms["eventCreateForm"]["expectednum"].value),
      "number_of_hosts": 0,
      "hosts": "",
      "hosting_organization": document.forms["eventCreateForm"]["hostingorg"].value,
      "organizer_id": organizer_id,
      "organizer_netid": String(Cookies.get('netid'))};

    const res = await fetch(database_url + '/event', {
        method: "POST",
        headers: {
            'Accept': 'application/json',
            "Content-Type": "application/json"
        },
        body: JSON.stringify(eventInfo)
    });

    this.toggle();
    this.refreshPage();
  }
  handleFiles = files => {
    var reader = new FileReader();
    var result = [];
    reader.onload = function(e) {
    // Use reader.result
    result = reader.result.split("\n")

    alert(result[1])
    }
    this.setState(state => ({ visitorEmails: result}));
  reader.readAsText(files[0]);
}
  toggle() {
    this.setState(prevState => ({
      modal: !prevState.modal
    }));
    if (!this.state.modal)
      this.setState(state => ({ visitorEmails: ""}));
  }
  async verifyUser(){
    var verified = false;
    var netid = String(Cookies.get('netid'));


    const session_current_organizer = await fetch(database_url + '/event_organizer/netidVerify/' + netid, {
            method: "GET",
            headers: {
              "Content-Type": "text/plain",
              "Access-Control-Allow-Origin": "*"
    }});

    var resp = await session_current_organizer.json();
    resp = JSON.stringify(resp);


    if (String(resp) === "{}")
    {
      //verified = true;
      //this.setState(state => ({ user_verified: true}));
      Router.push('/eventOrganizerRegister')

    }

  }
  static async getInitialProps({req}){

    var verified = false;
    var netid = String(Cookies.get('netid'));


    //console.log(netid);

    const session_current_organizer = await fetch(database_url + '/event_organizer/netidVerify/' + netid, {
            method: "GET",
            headers: {
              "Content-Type": "text/plain",
              "Access-Control-Allow-Origin": "*"

    }});

    var resp = await session_current_organizer.json();
    resp = JSON.stringify(resp);

    console.log(resp);

    if (String(resp) !== "{}")
    {
      verified = true;
      //this.setState(state => ({ user_verified: true}));
    }

    const res1 = await axios({
        url: server_url + '/netid',
        // manually copy cookie on server,
        // let browser handle it automatically on client
        headers: req ? {cookie: req.headers.cookie} : undefined,
      });


    const res2 = await fetch(database_url + '/event/organizersEvents/netid/' + res1.data.netid, {
            method: "GET",
            headers: {
                "Content-Type": "text/plain",
                "Access-Control-Allow-Origin": "*"
            }})
    var data = await res2.json()
    data = JSON.stringify(data)

    data = JSON.parse(data)


     //console.log(data)

     let descriptions = []
     let end_dates = []
     let end_times = []
     let expected_number_visitors = []
     let hosting_organizations = []
     let locations = []
     let names = []
     let number_of_hosts = []
     let start_dates = []
     let start_times = []
     let eventsHostSignup = []
     let eventsVisitorSignup = []
     let eventsClosedSignup = []
     let resArray = []

     for (let i = 0; i < data.length; i++)
     {
      // events.push(JSON.stringify(data[i]))
      if (data[i]['event_stage'] == 0)
      {
        eventsHostSignup.push(JSON.stringify(data[i]))
      }
      else if (data[i]['event_stage'] == 1)
      {
        eventsVisitorSignup.push(JSON.stringify(data[i]))
      }
      else
      {
        eventsClosedSignup.push(JSON.stringify(data[i]))
      }
     }

     return {
      descriptions: descriptions,
      end_dates: end_dates,
      end_times: end_times,
      expected_number_visitors: expected_number_visitors,
      hosting_organizations: hosting_organizations,
      locations: locations,
      names: names,
      number_of_hosts: number_of_hosts,
      start_dates: start_dates,
      start_times: start_times,
      eventsHostSignup:eventsHostSignup,
      eventsVisitorSignup:eventsVisitorSignup,
      eventsClosedSignup:eventsClosedSignup,
      userVerified: verified
     }

  }
  render(props){

      this.verifyUser();
      return(


      <div>
      <Head title="My Events" />

        <NavBar />

        <div className="hero">
          <center> <h2 style={divStyle}> My Events </h2> </center>
          <Button color="light" onClick={this.toggle}> <a className="text-dark"> Create a new event </a> </Button>
          <br />

      <Modal key="1" isOpen={this.state.modal} toggle={this.toggle} className={this.props.className}>
```

```
          <ModalHeader toggle={this.toggle} color="primary"> <p className="text-primary">Create an
Event</p></ModalHeader>
          <ModalBody>
          <Form id="eventCreateForm">
          <Row> <Col> Event Name: </Col> <Col> <Input type="text" name="eventname" id="eventname"/> </Col> </Row>
          <br />
          <Row>
          <Col>
          Start Date
          </Col>
          <Col>
          <Input type="date" name="startdate" id="startdate"/>
          </Col>
          <Col>
          Start Time
          </Col>
          <Col>
          <Input type="time" name="starttime" id="starttime"/>
          </Col>
          </Row>
          <br />
          <Row>
          <Col>
         End Date
          </Col>
          <Col>
          <Input type="date" name="enddate" id="enddate"/>
          </Col>
          <Col>
          End Time
          </Col>
          <Col>
          <Input type="time" name="endtime" id="endtime"/>
          </Col>
          </Row>
          <br />
          <Row>
          <Col>
          Expected Attendance
          </Col>
          <Col>
          <Input type="text" name="expectednum" id="expectednum"/>
          </Col>
          <Col> Location </Col>
          <Col> <Input type="text" name="location" id="location"/> </Col>
          </Row>
          <br />
          <Row>
          <Col>
          Hosting Organization
          </Col>
          <Col>
          <Input type="text" name="hostingorg" id="hostingorg"/>
          </Col>
          </Row>
          <br />
          Please add a brief description of the event: <Input type="textarea" name="description" id="description" />
          <br />
          </Form>

          </ModalBody>
          <ModalFooter>
            <Button color="primary" onClick={this.addEvent}>Create Event</Button>{' '}
            <Button color="secondary" onClick={this.toggle}>Cancel</Button>
          </ModalFooter>
      </Modal>
```

```
   <Modal key="2" isOpen={this.state.editModal} toggle={this.editModalToggle} className={this.props.className}>
      <ModalHeader toggle={this.editModalToggle}> <p className="text-primary"> Edit {this.state.current_event.name}
</p></ModalHeader>
      <ModalBody>
      <Form id="eventEditForm">
      <Row> <Col> Event Name: </Col> <Col> <Input type="text" name="eventname" id="eventname"
defaultValue={this.state.current_event.name}/> </Col> </Row>
      <br />
      <Row>
      <Col>
      Start Date
      </Col>
      <Col>
      <Input type="date" name="startdate" id="startdate" defaultValue={this.state.current_event.start_date}/>
      </Col>
      <Col>
      Start Time
      </Col>
      <Col>
      <Input type="time" name="starttime" id="starttime" defaultValue={this.state.current_event.start_time}/>
      </Col>
      </Row>
      <br />
      <Row>
      <Col>
     End Date
      </Col>
      <Col>
      <Input type="date" name="enddate" id="enddate" defaultValue={this.state.current_event.end_date}/>
      </Col>
      <Col>
      End Time
      </Col>
      <Col>
      <Input type="time" name="endtime" id="endtime" defaultValue={this.state.current_event.end_time}/>
      </Col>
      </Row>
      <br />
      <Row>
      <Col>
      Expected Attendance
      </Col>
      <Col>
      <Input type="text" name="expectednum" id="expectednum"
defaultValue={this.state.current_event.expected_number_visitors}/>
      </Col>
      <Col> Location </Col>
      <Col> <Input type="text" name="location" id="location" defaultValue={this.state.current_event.location}/> </Col>
      </Row>
      <br />
      <Row>
      <Col>
      Hosting Organization
      </Col>
      <Col>
      <Input type="text" name="hostingorg" id="hostingorg"
defaultValue={this.state.current_event.hosting_organization}/>
      </Col>
      </Row>
      <br />
      Please add a brief description of the event: <Input type="textarea" name="description" id="description"
defaultValue={this.state.current_event.description}/>
      <br />
      </Form>
```

```
          </ModalBody>
          <ModalFooter>
            <Button color="primary" onClick={this.editEvent}>Update Event</Button>{' '}
            <Button color="secondary" onClick={this.editModalToggle}>Cancel</Button>
          </ModalFooter>
        </Modal>

        <Modal key="3" isOpen={this.state.deleteModal} toggle={this.deleteModalToggle}
className={this.props.className}>
          <ModalHeader toggle={this.deleteModalToggle}> <p className="text-danger"> Delete
{this.state.current_event.name} </p></ModalHeader>
          <ModalBody>
          Are you sure you want to delete this event?
          </ModalBody>
          <ModalFooter>
            <Button color="danger" onClick={this.deleteEvent}>Delete Event</Button>{' '}
            <Button color="secondary" onClick={this.deleteModalToggle}>Cancel</Button>
          </ModalFooter>
        </Modal>
        <Modal key="4" isOpen={this.state.visitorSignupModal} toggle={this.visitorSignupToggle}
className={this.props.className}>
          <ModalHeader toggle={this.visitorSignupToggle}> <p className="text-success"> Begin Visitor Signup for
{this.state.current_event.name} </p></ModalHeader>
          <ModalBody>
          Are you sure you want to begin visitor signup for this event?
          </ModalBody>
          <ModalFooter>
            <Button color="success" onClick={this.beginVisitorSignup}>Begin Visitor Signup</Button>{' '}
            <Button color="secondary" onClick={this.visitorSignupToggle}>Cancel</Button>
          </ModalFooter>
        </Modal>

        <Modal key="5" isOpen={this.state.finalizePairingsModal} toggle={this.finalizePairingsToggle}
className={this.props.className}>
          <ModalHeader toggle={this.finalizePairingsToggle}> <p className="text-success"> Finalize Pairings for
{this.state.current_event.name} </p></ModalHeader>
          <ModalBody>
          Are you sure you want to finalize pairings for this event? Hosts and visitors will no longer be able to sign up.
          </ModalBody>
          <ModalFooter>
            <Button color="success" onClick={this.finalizePairings}>Finalize Pairings</Button>{' '}
            <Button color="secondary" onClick={this.finalizePairingsToggle}>Cancel</Button>
          </ModalFooter>
        </Modal>

        <Modal key="6" isOpen={this.state.viewDetailsModal} toggle={this.viewDetailsToggle}
className={this.props.className}>
          <ModalHeader toggle={this.viewDetailsToggle}> <p className="text-success"> View Pairing Details for
{this.state.current_event.name} </p></ModalHeader>
          <ModalBody>
         {this.state.pairings_for_event}
          </ModalBody>
          <ModalFooter>
            <Button color="secondary" onClick={this.viewDetailsToggle}>Exit</Button>
          </ModalFooter>
        </Modal>

        <br />

         <Nav tabs>
          <NavItem>
            <NavLink
              active={this.state.activeTab === '1'}
              onClick={() => { this.toggleTab('1'); }}
            >
              Pending
            </NavLink>
          </NavItem>
          <NavItem>
            <NavLink
              active={this.state.activeTab === '2'}
              onClick={() => { this.toggleTab('2'); }}
            >
              Visitor Signup in Progress
            </NavLink>
          </NavItem>
          <NavItem>
            <NavLink
              active={this.state.activeTab === '3'}
              onClick={() => { this.toggleTab('3'); }}
            >
              Completed
            </NavLink>
          </NavItem>
      </Nav>
      <br />

      <TabContent activeTab={this.state.activeTab}>
      <TabPane tabId="1">
        <CardDeck>
          {this.props.eventsHostSignup.map((value, index) => {
          let jsonVal = JSON.parse(value)
          return <div key={index}>
                 <Card className="card bg-light mb-3" key={index}>
                  <CardHeader key="0"> <center> <a style={divStyle2}> {jsonVal['name']} </a> </center>  </CardHeader>
                <img width="380" height="170" src="/static/conference.jpg" alt="Card image cap" />
                 {/* <p key="1"> Hosting Organization: {jsonVal['hosting_organization']} </p>
                 <p key="2"> Start Date: {jsonVal['start_date']} </p>
                 <p key="3"> Start Time: {jsonVal['start_time']} </p>
                 <p key="4"> End Date: {jsonVal['end_date']} </p>
                 <p key="5"> End Time: {jsonVal['end_time']} </p>
                 <p key="6"> Location: {jsonVal['location']} </p>  */}

                <Row>
                 <Col> <Button color="danger" key="10" size="sm" value={jsonVal['event_id']}
onClick={this.deleteModalToggle}> Cancel Event </Button> </Col>
                 <Col> <Button color="primary" key="8" size="sm" value={jsonVal['event_id']} onClick={this.editModalToggle}>
Edit Event </Button> </Col>
                 <Col> <Button color="success" key="9" size="sm" value={jsonVal['event_id']}
onClick={this.visitorSignupToggle}> Begin Visitor Signup</Button> </Col>

                 </Row>

                 </Card>
                 </div>

          })}
          </CardDeck>
          </TabPane>

          <TabPane tabId="2">
        <CardDeck>
          {this.props.eventsVisitorSignup.map((value, index) => {
          let jsonVal = JSON.parse(value)
          return <div key={index}>
                 <Card className="card bg-light mb-3" key={index}>
                  <CardHeader key="0"> <center> <a style={divStyle2}> {jsonVal['name']} </a> </center> </CardHeader>
                <img width="380" height="170" src="/static/conference.jpg" alt="Card image cap" />
                 {/* <p key="1"> Hosting Organization: {jsonVal['hosting_organization']} </p>
                 <p key="2"> Start Date: {jsonVal['start_date']} </p>
                 <p key="3"> Start Time: {jsonVal['start_time']} </p>
                 <p key="4"> End Date: {jsonVal['end_date']} </p>
                 <p key="5"> End Time: {jsonVal['end_time']} </p>
                 <p key="6"> Location: {jsonVal['location']} </p>  */}

                <Row>
                 <Col> <Button color="danger" key="10" size="sm" value={jsonVal['event_id']}
onClick={this.deleteModalToggle}> Cancel Event </Button> </Col>
                 <Col> <Button color="primary" key="8" size="sm" value={jsonVal['event_id']} onClick={this.editModalToggle}>
Edit Event </Button> </Col>
                 <Col> <Button color="success" key="9" size="sm" value={jsonVal['event_id']}
onClick={this.finalizePairingsToggle}> Finalize Pairings</Button> </Col>

                 </Row>

                 </Card>
                 </div>

          })}
          </CardDeck>
          </TabPane>
         <TabPane tabId="3">
        <CardDeck>
          {this.props.eventsClosedSignup.map((value, index) => {
          let jsonVal = JSON.parse(value)
          return <div key={index}>
                 <Card className="card bg-light mb-3" key={index}>
                  <CardHeader key="0"> <center> <a style={divStyle2}> {jsonVal['name']} </a> </center>  </CardHeader>
                <img width="365" height="170" src="/static/conference.jpg" alt="Card image cap" />
                 {/* <p key="1"> Hosting Organization: {jsonVal['hosting_organization']} </p>
                 <p key="2"> Start Date: {jsonVal['start_date']} </p>
                 <p key="3"> Start Time: {jsonVal['start_time']} </p>
                 <p key="4"> End Date: {jsonVal['end_date']} </p>
                 <p key="5"> End Time: {jsonVal['end_time']} </p>
                 <p key="6"> Location: {jsonVal['location']} </p>  */}

                <Row>
                 <Col> <Button color="danger" key="10" size="sm" value={jsonVal['event_id']}
onClick={this.deleteModalToggle}> Cancel Event </Button> </Col>
                 <Col> <Button color="primary" key="8" size="sm" value={jsonVal['event_id']} onClick={this.editModalToggle}>
Edit Event </Button> </Col>
                 <Col> <Button color="success" key="9" size="sm" value={jsonVal['event_id']} onClick={this.viewDetailsToggle}>
View Matchings </Button> </Col>

                 </Row>

                 </Card>
                 </div>

          })}
          </CardDeck>
          </TabPane>

        </TabContent>
      </div>
<style jsx>{`
        .hero {
          width: 100%;
          color: #333;
        }
        .title {
          margin: 0;
          width: 100%;
          padding-top: 80px;
          line-height: 1.15;
          font-size: 48px;
        }
        .title,
        .description {
          text-align: center;
        }
        .row {
          max-width: 880px;
          margin: 80px auto 40px;
          display: flex;
          flex-direction: row;
          justify-content: space-around;
        }
        .card {
          padding: 18px 18px 24px;
          width: 220px;
          text-align: left;
          text-decoration: none;
          color: #434343;
          border: 1px solid #9b9b9b;
        }
        .card:hover {
          border-color: #067df7;
        }
        .card h3 {
          margin: 0;
          color: #067df7;
          font-size: 18px;
        }
        .card p {
          margin: 0;
          padding: 12px 0 0;
          font-size: 13px;
          color: #333;
        }
      `}</style>
     </div>

)}}


export default eventList
```

hostAllEvents.js

```
import './bootstrap.css';
import React from 'react'
import Link from 'next/link'
import Head from '../components/head'
import HostNav from '../components/hostNav'
import fetch from 'isomorphic-unfetch'
import { Button, Modal, ModalHeader, ModalBody, ModalFooter, Row, Col, Input, Form, FormText, CustomInput } from
'reactstrap';
import { Card, CardImg, CardHeader, CardText, CardBody, CardTitle, CardSubtitle, CardDeck, FormGroup, Label} from
'reactstrap';
import ReactFileReader from 'react-file-reader';
import Cookies from 'js-cookie';

import Router from 'next/router'

const database_url = "https://tigernest-backend.herokuapp.com"
const server_url = "https://tiger-nest2.herokuapp.com"



var divStyle = {
  color: 'white'
  //color: 'dodgerblue'
};
```

```
var divStyle1 = {
  color: 'black'
  //color: 'dodgerblue'
};

const axios = require('axios');

class eventListHost extends React.Component {
  constructor(props, context){
    super(props, context);
    this.state = {
      modal: false,
      addHostModal: false,
      visitorEmails: [],
      current_event: {
        name:"",
        start_time:"",
        end_time:"",
        description:"",
        hosting_organization:"",
        expected_number_visitors:"",
        event_id:""
       }
    };
    this.addHostToggle = this.addHostToggle.bind(this);
    this.addHost = this.addHost.bind(this);
    this.editEvent = this.editEvent.bind(this);
    this.handleFiles = this.handleFiles.bind(this);
    this.refreshPage = this.refreshPage.bind(this);
  }
  refreshPage(){
    window.location.reload();
  }
  async editEvent(){
    console.log(this.state.visitorEmails)

    let eventInfo = {
      "name": document.forms["eventEditForm"]["eventname"].value,
      "start_time": document.forms["eventEditForm"]["starttime"].value,
      "start_date": document.forms["eventEditForm"]["startdate"].value,
      "end_time": document.forms["eventEditForm"]["endtime"].value,
      "end_date": document.forms["eventEditForm"]["enddate"].value,
      "description": document.forms["eventEditForm"]["description"].value,
      "location": document.forms["eventEditForm"]["location"].value,
      "expected_number_visitors": parseInt(document.forms["eventEditForm"]["expectednum"].value),
      "number_of_hosts": 0,
      "hosts": "",
      "hosting_organization": document.forms["eventEditForm"]["hostingorg"].value,
      "organizer_id": 1,};

    const res = await fetch(database_url + '/event/update/' + this.state.current_event.event_id, {
        method: "PUT",
        headers: {
            'Accept': 'application/json',
            "Content-Type": "application/json"
        },
        body: JSON.stringify(eventInfo)
    });

    this.editModalToggle();
    this.refreshPage();


  }
  async addHostToggle(event) {
    this.setState(state => ({ visitorEmails: ""}));
    //this.setState(state => ({ current_event: event.target.value}));
    if (!this.state.addHostModal)
    {
      let val = event.target.value;

      const res = await fetch(database_url + '/event/' + val, {
           method: "GET",
           headers: {
               "Content-Type": "text/plain",
               "Access-Control-Allow-Origin": "*"
        }})
      var data = await res.json()
      data = JSON.stringify(data)
      data = JSON.parse(data)
      this.setState(state => ({ current_event: data}));
    }

      //console.log(data);


    this.setState(prevState => ({
      addHostModal: !prevState.addHostModal
    }));



  }
  async addHost(){

    let genderVal = ""
    let host_gender = ""

    try {
      genderVal = document.forms["hostSignupForm"]["radio1"].value;
      host_gender = document.forms["hostSignupForm"]["radio2"].value;
    }
    catch(err) {
      alert("All fields are required");
      return;
    }

    if (document.forms["hostSignupForm"]["firstname"].value === "" || document.forms["hostSignupForm"]["lastname"].value
=== "" || document.forms["hostSignupForm"]["roomnum"].value === "" ||
document.forms["hostSignupForm"]["maxvisitors"].value === "" || document.forms["hostSignupForm"]["cellnum"].value ===
"")
    {
      alert("All fields are required");
      return;
    }

    if (isNaN(parseInt((document.forms["hostSignupForm"]["maxvisitors"].value))))
    {
      alert("Max Visitors must be an integer!");
      return;
    }

    let same_gender = genderVal === "yes" ? false : true;
    //console.log(this.state.visitorEmails)
    let pairingInfo = {
      "host_first_name": document.forms["hostSignupForm"]["firstname"].value,
      "host_last_name": document.forms["hostSignupForm"]["lastname"].value,
      "host_netid": String(Cookies.get('netid')),
      "host_gender": host_gender,
      "same_gender_room": same_gender,
      "host_room_num": document.forms["hostSignupForm"]["roomnum"].value,
      "max_visitors": document.forms["hostSignupForm"]["maxvisitors"].value,
      "visitor_list": {},
      "host_cellphone": document.forms["hostSignupForm"]["cellnum"].value,
      "event_id": this.state.current_event.event_id,
      "event_name": this.state.current_event.name
      };

    const res = await fetch(database_url + '/pairing', {
        method: "POST",
        headers: {
            'Accept': 'application/json',
            "Content-Type": "application/json"
        },
        body: JSON.stringify(pairingInfo)
    });


    this.addHostToggle();
    this.refreshPage();
    //this.getInitialProps();
  }
  handleFiles = files => {
    var reader = new FileReader();
    var result = [];
    reader.onload = function(e) {
    // Use reader.result
    result = reader.result.split(",")

    alert(result[0])
    }
    this.setState(state => ({ visitorEmails: result}));
  reader.readAsText(files[0]);
}
  toggle() {
    this.setState(prevState => ({
      modal: !prevState.modal
    }));
    if (!this.state.modal)
      this.setState(state => ({ visitorEmails: ""}));
  }
  static async getInitialProps({req}){

    //const res1 = await fetch('http://localhost:5000/')

    const res1 = await axios({
        url: server_url + '/netid',
        // manually copy cookie on server,
        // let browser handle it automatically on client
        headers: req ? {cookie: req.headers.cookie} : undefined,
      });
    const res2 = await fetch(database_url + '/pairing/events_for_host/' + res1.data.netid, {
            method: "GET",
            headers: {
                "Content-Type": "text/plain",
                "Access-Control-Allow-Origin": "*"
            }})

    var data2 = await res2.json()
      data2 = JSON.stringify(data2)

     data2 = JSON.parse(data2)

     let eventsHost = []

     for(let i = 0; i < data2.length; i++)
     {
```

```
  eventsHost.push(data2[i]['event_id'])
 }


//console.log(res1.data)


    const res = await fetch(database_url + '/event/sort_date', {
        method: "GET",
        headers: {
            "Content-Type": "text/plain",
            "Access-Control-Allow-Origin": "*"
        }})
  var data = await res.json()
  data = JSON.stringify(data)

 data = JSON.parse(data)
 //console.log(data)

 /*
    Get all the events that the host is in, put this in a table.
    Make sure that they do not sign up twice for the same event.
    Events that they already signed up for should have a "drop event" option.

 */

 let descriptions = []
 let end_dates = []
 let end_times = []
 let expected_number_visitors = []
 let hosting_organizations = []
 let locations = []
 let names = []
 let number_of_hosts = []
 let start_dates = []
 let start_times = []
 let events = []

 for (let i = 0; i < data.length; i++)
 {
  if (!eventsHost.includes(data[i]['event_id']) && (data[i]['event_stage'] == 0 || data[i]['event_stage'] == 1))
    events.push(JSON.stringify(data[i]));
 }

 return {
  descriptions: descriptions,
  end_dates: end_dates,
  end_times: end_times,
  expected_number_visitors: expected_number_visitors,
  hosting_organizations: hosting_organizations,
  locations: locations,
  names: names,
  number_of_hosts: number_of_hosts,
  start_dates: start_dates,
  start_times: start_times,
  events:events,
  data: res1.data
 }

  /*return {
    description: data['description'],
    end_date: data['end_date'],
    end_time: data['end_time'],
    expected_number_visitors: data['expected_number_visitors'],
    hosting_organization: data['hosting_organization'],
    location: data['location'],
        name: data['name'],
        number_of_hosts: data['number_of_hosts'],
        start_date: data['start_date'],
        start_time: data['start_time']
      } */


  }
  render(props){
    //console.log(this.props.data);
    return(
  <div>
  <Head title="Host Events" />
    <HostNav />

    <div className="hero">
      <center> <h2 style={divStyle}> Host for an event!</h2> </center>
      <br />

      <Modal key="1" isOpen={this.state.addHostModal} toggle={this.addHostToggle} className={this.props.className}>
          <ModalHeader toggle={this.addHostToggle}> <p className="text-primary"> Host for
{this.state.current_event.name} </p> </ModalHeader>
          <ModalBody>
          <Form id="hostSignupForm">
          <Row>
          <Col>
          First Name
          </Col>
          <Col>
          <Input type="text" name="firstname" id="firstname"/>
          </Col>
          </Row>
          <br />
          <Row>
          <Col>
          Last Name
          </Col>
          <Col>
          <Input type="text" name="lastname" id="lastname"/>
          </Col>
          </Row>
          <br />
          <Row>
          <Col>
          Room Number (Ex: Little Hall 99)
          </Col>
          <Col>
          <Input type="text" name="roomnum" id="roomnum"/>
          </Col>
          </Row>
          <br />
          <Row>
          <Col>
          Cellphone Number:
          </Col>
          <Col>
          <Input type="text" name="cellnum" id="cellnum"/>
          </Col>
          </Row>
          <br />
          <Row>
          <Col>
          Max Number of Visitors:
          </Col>
          <Col>
          <Input type="text" name="maxvisitors" id="maxvisitors"/>
          </Col>
          </Row>
          <br />
          <Row>
          <Col> Gender </Col>
          <Col>
          <FormGroup check>
              <Label check>
                <Input type="radio" name="radio2" id="radio2" value="Male"/>{' '}
                Male
              </Label>
            </FormGroup>
            <FormGroup check>
              <Label check>
                <Input type="radio" name="radio2" id="radio2" value="Female"/>{' '}
                Female
              </Label>
            </FormGroup>
           </Col>
          </Row>
          <br />
          <Row>
          <Col> Opposite Gender Visitors</Col>
          <Col>
          <FormGroup check>
              <Label check>
                <Input type="radio" name="radio1" id="radio1" value="yes"/>{' '}
                Yes
              </Label>
            </FormGroup>
            <FormGroup check>
              <Label check>
                <Input type="radio" name="radio1" id="radio1" value="no"/>{' '}
                No
              </Label>
            </FormGroup>
          </Col>
          </Row>
          <br />
          </Form>
          </ModalBody>
          <ModalFooter>
            <Button color="primary" onClick={this.addHost}> Sign Up</Button>{' '}
            <Button color="secondary" onClick={this.addHostToggle}>Cancel</Button>
          </ModalFooter>
      </Modal>



      <CardDeck>
        {this.props.events.map((value, index) => {
          let jsonVal = JSON.parse(value)
          return <div key={index}>
          <Card className="card bg-light mb-3" key={index}>
                  <CardHeader key="0"> <center> <a style={divStyle1}> {jsonVal['name']} </a> </center> </CardHeader>
                <img width="350" height="170" src="/static/conference.jpg" alt="Card image cap" />

                {/*  <p key="1"> Hosting Organization: {jsonVal['hosting_organization']} </p>
                 <p key="2"> Start Date: {jsonVal['start_date']} </p>
                 <p key="3"> Start Time: {jsonVal['start_time']} </p>
                 <p key="4"> End Date: {jsonVal['end_date']} </p>
                 <p key="5"> End Time: {jsonVal['end_time']} </p>
                 <p key="6"> Location: {jsonVal['location']} </p>    */}
                 <center>  <p key="2"> {jsonVal['start_date']} to {jsonVal['end_date']}  </p> </center>
                 <center>  <p key="3"> {jsonVal['location']}  </p> </center>
                <center> <p key="4">  {jsonVal['start_time']}  </p> </center>
```

```
                <Button color="primary" key="8" value={jsonVal['event_id']} onClick={this.addHostToggle}> Sign up to host
</Button>
                </Card>
                </div>

        })}
        </CardDeck>
    </div>
<style jsx>{`
      .hero {
        width: 100%;
        color: #333;
      }
      .title {
        margin: 0;
        width: 100%;
        padding-top: 80px;
        line-height: 1.15;
        font-size: 48px;
      }
      .title,
      .description {
        text-align: center;
      }
      .row {
        max-width: 880px;
        margin: 80px auto 40px;
        display: flex;
        flex-direction: row;
        justify-content: space-around;
      }
      .card {
        padding: 30px 30px 30px;
        width: 300px;
        text-align: left;
        text-decoration: none;
        color: #434343;
        border: 1px solid #9b9b9b;
      }
      .card:hover {
        border-color: #067df7;
      }
      .card h3 {
        margin: 0;
        color: #067df7;
        font-size: 18px;
      }
      .card p {
        margin: 0;
        padding: 12px 0 0;
        font-size: 13px;
        color: #333;
      }
    `}</style>
  </div>

)}
}

export default eventListHost
```

hostMyEvents.js

```
import './bootstrap.css';
import React from 'react'
import Link from 'next/link'
import Head from '../components/head'
import HostNav from '../components/hostNav'
import fetch from 'isomorphic-unfetch'
import { Button, Modal, ModalHeader, ModalBody, ModalFooter, Row, Col, Input, Form, FormText, CustomInput } from
'reactstrap';
import { Card, CardImg, CardText, CardBody, CardTitle, CardSubtitle, CardDeck, FormGroup, Label, Table} from
'reactstrap';
import ReactFileReader from 'react-file-reader';
import Cookies from 'js-cookie';

import Router from 'next/router'

//const database_url = "http://localhost:5000"
const database_url = "https://tigernest-backend.herokuapp.com"

//const server_url = "http://localhost:3000"
const server_url = "https://tiger-nest2.herokuapp.com"

var divStyle = {
  color: 'white'
  //color: 'dodgerblue'
};

const axios = require('axios');

class eventListHost extends React.Component {
  constructor(props, context){
    super(props, context);
    this.state = {
      modal: false,
      dropEventModal: false,
      addHostModal: false,
      visitorEmails: [],
      current_pairing: -1,
      eventList: [],
      viewDetailsModal: false,
      visitor_list: [],
      male_true: false,
      female_true: false,
      same_gender_yes: false,
      same_gender_no: false,
      current_event_name: "",
      current_event: {
        name:"",
        start_time:"",
        end_time:"",
        description:"",
        hosting_organization:"",
        expected_number_visitors:"",
        event_id:""
      }
    };
    this.addHostToggle = this.addHostToggle.bind(this);
    this.addHost = this.addHost.bind(this);
    this.editEvent = this.editEvent.bind(this);
    this.handleFiles = this.handleFiles.bind(this);
    this.dropEventToggle = this.dropEventToggle.bind(this);
    this.dropEvent = this.dropEvent.bind(this);
    this.filterEvents = this.filterEvents.bind(this);
    this.viewDetailsToggle = this.viewDetailsToggle.bind(this);
    this.refreshPage = this.refreshPage.bind(this);
```

```
}
refreshPage(){
  window.location.reload();
}
async viewDetailsToggle(){
  if (!this.state.viewDetailsModal)
  {
    let val = event.target.value;

    const res = await fetch(database_url + '/visitor_pairing/all_hosts/' + val, {
         method: "GET",
         headers: {
             "Content-Type": "text/plain",
             "Access-Control-Allow-Origin": "*"
    }})

    var data = await res.json()

    let visitornames = []

    for(let i = 0; i < data.length; i++)
    {
      let visitor_id = data[i]['visitor_id']
      let resVisitor = await fetch(database_url + '/visitor/' + visitor_id, {
         method: "GET",
         headers: {
             "Content-Type": "text/plain",
             "Access-Control-Allow-Origin": "*"
      }})
      var dataVisitor = await resVisitor.json()
      dataVisitor = JSON.stringify(dataVisitor)
      dataVisitor = JSON.parse(dataVisitor)
      visitornames.push(String(dataVisitor['name'] + "\n"))
    }

    //data = JSON.stringify(data);
    //data = JSON.parse(data);
    this.setState(state => ({ visitor_list: visitornames}));
  }

  this.setState(prevState => ({
    viewDetailsModal: !prevState.viewDetailsModal
  }));

}
async editEvent(){
  console.log(this.state.visitorEmails)

  let eventInfo = {
    "name": document.forms["eventEditForm"]["eventname"].value,
    "start_time": document.forms["eventEditForm"]["starttime"].value,
    "start_date": document.forms["eventEditForm"]["startdate"].value,
    "end_time": document.forms["eventEditForm"]["endtime"].value,
    "end_date": document.forms["eventEditForm"]["enddate"].value,
    "description": document.forms["eventEditForm"]["description"].value,
    "location": document.forms["eventEditForm"]["location"].value,
    "expected_number_visitors": parseInt(document.forms["eventEditForm"]["expectednum"].value),
    "number_of_hosts": 0,
    "hosts": "",
    "hosting_organization": document.forms["eventEditForm"]["hostingorg"].value,
    "organizer_id": 1,};

  const res = await fetch(database_url + '/event/update/' + this.state.current_event.event_id, {
      method: "POST",
      headers: {
          'Accept': 'application/json',
          "Content-Type": "application/json"
      },
      body: JSON.stringify(eventInfo)
  });

  this.editModalToggle();
  this.refreshPage();


}
async dropEventToggle(event) {

  //this.setState(state => ({ visitorEmails: ""}));
  //this.setState(state => ({ current_event: event.target.value}));
  if (!this.state.dropEventModal)
  {
    let val = event.target.value;

    const res = await fetch(database_url + '/pairing/' + val, {
        method: "GET",
        headers: {
            "Content-Type": "text/plain",
            "Access-Control-Allow-Origin": "*"
      }})
    var data = await res.json()
    data = JSON.stringify(data)
    data = JSON.parse(data)
    this.setState(state => ({ current_pairing: data}));
  }

    //console.log(data);


  this.setState(prevState => ({
    dropEventModal: !prevState.dropEventModal
  }));



}
async addHostToggle(event) {

  this.setState(state => ({ visitorEmails: ""}));
  //this.setState(state => ({ current_event: event.target.value}));
  if (!this.state.addHostModal)
  {
    let val = event.target.value;

    const res = await fetch(database_url + '/pairing/' + val, {
        method: "GET",
        headers: {
            "Content-Type": "text/plain",
            "Access-Control-Allow-Origin": "*"
      }})
    var data = await res.json()
    data = JSON.stringify(data)
    data = JSON.parse(data)
    this.setState(state => ({ current_pairing: data}));


    if (String(data['host_gender']) === "Male")
    {
      this.setState(state => ({ male_true: true}));
      this.setState(state => ({ female_true: false}));
    }
    else
```

```
      {
        this.setState(state => ({ female_true: true}));
        this.setState(state => ({ male_true: false}));
      }
      if ((data['same_gender_room']))
      {
        this.setState(state => ({ same_gender_yes: true}));
        this.setState(state => ({ same_gender_no: false}));
      }
      else
      {
        this.setState(state => ({ same_gender_no: true}));
        this.setState(state => ({ same_gender_yes: false}));
      }
    }

      //console.log(data);


    this.setState(prevState => ({
      addHostModal: !prevState.addHostModal
    }));



  }
  async dropEvent(){

     const res = await fetch(database_url + '/pairing/delete/' + this.state.current_pairing.pairing_id, {
        method: "DELETE",
        headers: {
            'Accept': 'application/json',
            "Content-Type": "application/json"
        }
    });

     this.dropEventToggle();
     this.refreshPage();
  }
  async addHost(){
    //let name = document.forms["eventCreateForm"]["eventname"].value;
    //console.log(name)

    let genderVal = ""
    let host_gender = ""

    try {
      genderVal = document.forms["hostSignupForm"]["radio1"].value;
      host_gender = document.forms["hostSignupForm"]["radio2"].value;
    }
    catch(err) {
      alert("All fields are required");
      return;
    }

    if (document.forms["hostSignupForm"]["firstname"].value === "" || document.forms["hostSignupForm"]["lastname"].value
=== "" || document.forms["hostSignupForm"]["roomnum"].value === "" ||
document.forms["hostSignupForm"]["maxvisitors"].value === "" || document.forms["hostSignupForm"]["cellnum"].value ===
"")
    {
      alert("All fields are required");
      return;
    }

    if (isNaN(parseInt((document.forms["hostSignupForm"]["maxvisitors"].value))))
    {
      alert("Max Visitors must be an integer!");
      return;
    }

    let same_gender = genderVal === "yes" ? true : false;
    //console.log(this.state.visitorEmails)
    let pairingInfo = {
      "host_first_name": document.forms["hostSignupForm"]["firstname"].value,
      "host_last_name": document.forms["hostSignupForm"]["lastname"].value,
      "host_gender": host_gender,
      "same_gender_room": same_gender,
      "host_room_num": document.forms["hostSignupForm"]["roomnum"].value,
      "max_visitors": document.forms["hostSignupForm"]["maxvisitors"].value,
      "host_cellphone": document.forms["hostSignupForm"]["cellnum"].value,
      "event_id": this.state.current_event.event_id
      };

    const res = await fetch(database_url + '/pairing/update/' + this.state.current_pairing.pairing_id, {
        method: "POST",
        headers: {
            'Accept': 'application/json',
            "Content-Type": "application/json"
        },
        body: JSON.stringify(pairingInfo)
    });


    this.addHostToggle();
    this.refreshPage();
  }
  handleFiles = files => {
    var reader = new FileReader();
    var result = [];
    reader.onload = function(e) {
    // Use reader.result
    result = reader.result.split(",")

    alert(result[0])
    }
    this.setState(state => ({ visitorEmails: result}));
  reader.readAsText(files[0]);
}
  toggle() {
    this.setState(prevState => ({
      modal: !prevState.modal
    }));
    if (!this.state.modal)
      this.setState(state => ({ visitorEmails: ""}));
  }
  async filterEvents()
  {
    var netid = String(Cookies.get('netid'));

    const res = await fetch(database_url + '/pairing/events_for_host/' + netid, {
            method: "GET",
            headers: {
                "Content-Type": "text/plain",
                "Access-Control-Allow-Origin": "*"
            }})
      var data = await res.json()
      data = JSON.stringify(data)

     data = JSON.parse(data)

     let events = []
```

```
   for (let i = 0; i < data.length; i++)
   {
    /*descriptions.push(data[i]['description']);
    end_dates.push(data[i]['end_dates'])
    end_times.push(data[i]['end_times'])
    expected_number_visitors.push(data[i]['expected_number_visitors'])
    hosting_organizations.push(data[i]['hosting_organizations'])
    locations.push(data[i]['locations'])
    names.push(data[i]['names'])
    number_of_hosts.push(data[i]['number_of_hosts'])
    start_dates.push(data[i]['start_date'])
    start_times.push(data[i]['start_time'])*/
    events.push(JSON.stringify(data[i]))
   }

   //console.log(events)

   this.setState({
     eventList: events
    });
 }
 static async getInitialProps({req}){
     //const netid = Cookies.get('netid')
     /*const res = await fetch('http://localhost:5000/pairing/events_for_host/' + netid, {
         method: "GET",
         headers: {
             "Content-Type": "text/plain",
             "Access-Control-Allow-Origin": "*"
         }})
    var data = await res.json()
    data = JSON.stringify(data)

   data = JSON.parse(data) */
   //console.log(data)

   const res1 = await axios({
     url: server_url + '/netid',
     // manually copy cookie on server,
     // let browser handle it automatically on client
     headers: req ? {cookie: req.headers.cookie} : undefined,
    });

   const res = await fetch(database_url + '/pairing/events_for_host/' + res1.data.netid, {
         method: "GET",
         headers: {
             "Content-Type": "text/plain",
             "Access-Control-Allow-Origin": "*"
         }})
    var data = await res.json()
    data = JSON.stringify(data)

   data = JSON.parse(data)

   let descriptions = []
   let end_dates = []
   let end_times = []
   let expected_number_visitors = []
   let hosting_organizations = []
   let locations = []
   let names = []
   let number_of_hosts = []
   let start_dates = []
   let start_times = []
   let events = []
```

```
    for(let i = 0; i < data.length; i++)
    {
     events.push(data[i]);
    }


    return {
     descriptions: descriptions,
     end_dates: end_dates,
     end_times: end_times,
     expected_number_visitors: expected_number_visitors,
     hosting_organizations: hosting_organizations,
     locations: locations,
     names: names,
     number_of_hosts: number_of_hosts,
     start_dates: start_dates,
     start_times: start_times,
     events: events
    }

     /*return {
       description: data['description'],
       end_date: data['end_date'],
       end_time: data['end_time'],
       expected_number_visitors: data['expected_number_visitors'],
       hosting_organization: data['hosting_organization'],
       location: data['location'],
       name: data['name'],
       number_of_hosts: data['number_of_hosts'],
       start_date: data['start_date'],
       start_time: data['start_time']
     } */


  }
  render(props){

    /*this.setState({
       eventList: this.filterEvents()
     }); */
     //this.filterEvents();
    return(
  <div>
  <Head title="Host Events" />
    <HostNav />

    <div className="hero">
     <center> <h2 style={divStyle}> Your Events</h2> </center>
     <br />


     <Modal key="1" isOpen={this.state.addHostModal} toggle={this.addHostToggle} className={this.props.className}>
         <ModalHeader toggle={this.addHostToggle}> <p className="text-primary"> Edit Information for
{this.state.current_pairing.event_name} </p> </ModalHeader>
         <ModalBody>
         <Form id="hostSignupForm">
         <Row>
         <Col>
         First Name:
         </Col>
         <Col>
         <Input type="text" name="firstname" id="firstname" defaultValue={this.state.current_pairing.host_first_name}/ >
         </Col>
         </Row>
         <br />
         <Row>
```

```
<Col>
Last Name:
</Col>
<Col>
<Input type="text" name="lastname" id="lastname" defaultValue={this.state.current_pairing.host_last_name}/>
</Col>
</Row>
<br />
<Row>
<Col>
Room Number:
</Col>
<Col>
<Input type="text" name="roomnum" id="roomnum" defaultValue={this.state.current_pairing.host_room_num}/>
</Col>
</Row>
<br />
<Row>
<Col>
Cellphone Number:
</Col>
<Col>
<Input type="text" name="cellnum" id="cellnum" defaultValue={this.state.current_pairing.host_cellphone}/>
</Col>
</Row>
<br />
<Row>
<Col>
Max Number of Visitors:
</Col>
<Col>
<Input type="text" name="maxvisitors" id="maxvisitors" defaultValue={this.state.current_pairing.max_visitors}/>
</Col>
</Row>
<br />
<Row>
<Col> Gender </Col>
<Col>
<FormGroup check>
    <Label check>
      <Input type="radio" name="radio2" id="radio2" value="Male" defaultChecked={this.state.male_true} />{' '}
      Male
    </Label>
  </FormGroup>
  <FormGroup check>
    <Label check>
      <Input type="radio" name="radio2" id="radio2" value="Female" defaultChecked={this.state.female_true} />{' '}
      Female
    </Label>
  </FormGroup>
 </Col>
</Row>
<br />
<Row>
<Col> Opposite Gender Visitors</Col>
<Col>
<FormGroup check>
    <Label check>
      <Input type="radio" name="radio1" id="radio1" value="yes" defaultChecked={this.state.same_gender_yes} />{' '}
      Yes
    </Label>
  </FormGroup>
  <FormGroup check>
    <Label check>
      <Input type="radio" name="radio1" id="radio1" value="no" defaultChecked={this.state.same_gender_no} />{' '}
      No
```

```
            </Label>
          </FormGroup>
        </Col>
        </Row>
        <br />
        </Form>
        </ModalBody>
        <ModalFooter>
          <Button color="primary" onClick={this.addHost}> Update Info </Button>{' '}
          <Button color="secondary" onClick={this.addHostToggle}>Cancel</Button>
        </ModalFooter>
    </Modal>

     <Modal key="2" isOpen={this.state.dropEventModal} toggle={this.dropEventToggle}
className={this.props.className}>
        <ModalHeader toggle={this.dropEventToggle}> <p className="text-danger"> Drop
{this.state.current_pairing.event_name} </p> </ModalHeader>
        <ModalBody>

        Are you sure you want to drop out of this event?

        </ModalBody>

        <ModalFooter>
          <Button color="danger" onClick={this.dropEvent}> Drop Event</Button>{' '}
          <Button color="secondary" onClick={this.dropEventToggle}>Cancel</Button>
        </ModalFooter>
      </Modal>

       <Modal key="6" isOpen={this.state.viewDetailsModal} toggle={this.viewDetailsToggle}
className={this.props.className}>
        <ModalHeader toggle={this.viewDetailsToggle}> <p className="text-success"> View Your Visitor List!
</p></ModalHeader>
        <ModalBody>
       {this.state.visitor_list}
        </ModalBody>
        <ModalFooter>
          <Button color="secondary" onClick={this.viewDetailsToggle}>Exit</Button>
        </ModalFooter>
    </Modal>


       <Table dark>
       <tbody>

       {this.props.events.map((value, index) => {
        //console.log("value" + value);
        let jsonVal = value;
        //let jsonVal = value;
        let tableHead = "";
        if (index == 0)
        {
          tableHead = "<thead> <tr> <th> Name </th> <th> your gender </th>"
        }
        return <div>
               <tr key={index}>
               <th key="0"> Event Name: {jsonVal['event_name']} </th>
               <th key="3"> Maximum Visitors: {jsonVal['max_visitors']} </th>
               <th key="6"> Cellphone: {jsonVal['host_cellphone']} </th>
               <br />
                <th> <Button color="light" key="8" value={jsonVal['pairing_id']} onClick={this.addHostToggle} size="sm" > Edit
Information </Button> </th>
               <th> <Button color="danger" key="9" value={jsonVal['pairing_id']} onClick={this.dropEventToggle} size="sm">
Drop Event </Button> </th>
               <th> <Button color="success" key="9" value={jsonVal['pairing_id']} onClick={this.viewDetailsToggle} size="sm">
View Visitors </Button> </th>
```

```
                </tr>
                </div>


        })}
         </tbody>
         </Table>
    </div>
<style jsx>{`
      .hero {
        width: 100%;
        color: #333;
      }
      .title {
        margin: 0;
        width: 100%;
        padding-top: 80px;
        line-height: 1.15;
        font-size: 48px;
      }
      .title,
      .description {
        text-align: center;
      }
      .row {
        max-width: 880px;
        margin: 80px auto 40px;
        display: flex;
        flex-direction: row;
        justify-content: space-around;
      }
      .card {
        padding: 30px 30px 30px;
        width: 300px;
        text-align: left;
        text-decoration: none;
        color: #434343;
        border: 1px solid #9b9b9b;
      }
      .card:hover {
        border-color: #067df7;
      }
      .card h3 {
        margin: 0;
        color: #067df7;
        font-size: 18px;
      }
      .card p {
        margin: 0;
        padding: 12px 0 0;
        font-size: 13px;
        color: #333;
      }
    `}</style>
  </div>

)}
}

export default eventListHost
```