**E-Spotter**

**Mario Orbegoso, Graciany Lebrón, and Carlos Ojeda**

**mario.orbegoso@upr.edu, graciany.lebron@upr.edu, carlos.ojeda4@upr.edu**

**1.** **Introduction**

Currently, there are many student associations in the University of Puerto Rico at Mayaguez that have different activities every day. Nowadays, the most common way to learn about these activities is by either checking your emails sent by “el cartero” to see if you have received information about the activity or by checking the Facebook page of each corresponding association. This method is not efficient because students receive multiple emails every day and will probably skip over the emails announcing the associations’ activities. The other alternative is by checking Facebook for information about the event but you need to know the association’s name. Our product aims to ease these problems and help the student body by creating a web app that will consolidate and facilitate all of the associations’ announcements of their events and let the students track and discover events that they find attractive all in one place. On a more technical note, we will implement the front-end with HTML5, CSS3 and Javascript, using the view library ReactJS together with Redux and Bootstrap for styling. The server side application is going to be implemented in Node.js and the actual database will be managed with the relational database system called PostgreSQL.

**2.** **Client App Description**

There will be two ways to sign up in our app, as a regular user or as an official student association. Both signup processes require email verification, in the case of an association account there will be further verification to validate their legitimacy. The regular user will be able to view all of the events that registered associations have at the time being and also give a review of them. These events appear by most viewed, most recent, most interested and closest date depending on the tab that the user is currently viewing. In addition, he may also follow any of the associations in the app, this way the user will be informed immediately when their preferred associations will have a new event or if they have made any updates to their existing events only if they flagged it as “interesting”. As a result, it will appear in a special tab where they can view the events carefully since some events will require payment and attendance confirmation.

On the other side, the associations will take a more active role than that of a regular user since they are the ones who can create and post the events that the regular user will react to. The registered associations will have access to statistics regarding their event such as the number of users that are interested, user payment transactions and the amount of users who confirmed their attendance. Meanwhile they can also keep updating the events if any unpredicted changes occur prior to the activity.

There are various technologies used to create the front-end of an application. We will implement the front-end with HTML5, CSS3 and Javascript, using the view library ReactJS together with Redux and Bootstrap for styling to helps us develop all of these features that our users will interact with.

**3.** **Server Side Description**

The server side application is going to be implemented in Node.js which will hold all the business logic of the application. It will also provide a RESTful API that will respond to our client’s request. Furthermore, we will be using the relational database system called PostgreSQL to store and manage all the sensitive information of our application. For signup and account confirmation and to support payment in our application a third party API is most likely going to be used such as Paypal and SendGrid. The server will contain ten tables. These tables are Users, Associations, Live Events, Transactions, Statistics, Past Events, Sponsors and Faculties.

Table Descriptions:

1. Users - this table will include all the attributes that identifies the regular user in our system.
2. Associations - this table will include the list of associations that are registered in our system and it will include all the attributes that describe an association and will differentiate it from a regular user.
3. Events - this table will include all of the events with their start date and end date..
4. Transactions - this table will include all of the transactions that happened prior the event.
5. Sponsors - this table will have the list of companies that sponsor the associations.
6. Location - this table will have the location of the associations and the location of the events.
7. Reviews - this table will have the reviews that the user will write to rate the association.
8. Account - this table will have the email and password of all associations and users.
9. Image - this table will include all of the different images that a user, event and association can use.
10. Notifications - this table will include the different notification types that an association .
11. Reviews- this table will include the review that the regular user can post about an association.

**4.** **E-R Diagram (updated)**

Present your E-R diagram and explain it.



**ER Diagram Explanation:**

In our Entity Relationship diagram, we have 3 main entities that are the center of the application, these are the association users, the student users and the events themselves, these entities are the ones that make our app the way it is. The association and student user entities have similar relationships with the other entities the only difference is that the associations have more creative power overall since they are the one who actually create the events. There are relationships that are necessary to have full participation relationship and that is to have an account, without it any type of user will not be able to do anything within our application and also every user should have only one account. It is important to note that the events and associations have a many to one relationship respectively since we are forcing a constraint that two or more associations can’t host the same event for now. Overall the student user with its relationships with the other entities he or she can the following: they can write reviews to provide feedback of the events, follow various associations, be interested in different events, do only one payment within the Transaction entity which is the entrance fee, they can also use images for his or her profile picture, and receive notifications of their interested events and the associations they follow. On the other hand an Association can create the events, have sponsors, be followed by the students, receive and send notifications, have a location for their main offices, use images on their profile and when they post their events, and they also sell tickets for their events using the Transaction entity in other words the entrance fee. Finally the event entity should have their sponsors as well but there will be only place in the association’s profile, also the events can’t have various locations if they last more than one day that’s why we are forcing a one to many relationship with location and events respectively for now, the events can also receive various reviews from a user, have an image as their flyer, they can also be interested by the users, created by the associations, be divided in categories and have stats of the amount of people who are interested, their gender, their major, age and the university they attend . As a result, this is the overall dynamic that our application has due to the nature of the relationships that each entity has with each other.

**Create Table Dump Files**

**Account:**

CREATE TABLE public.account

(

account\_id integer NOT NULL DEFAULT nextval('account\_accountid\_seq'::regclass),

email character varying(35) COLLATE pg\_catalog."default" NOT NULL,

password character varying(20) COLLATE pg\_catalog."default" NOT NULL,

date\_created character varying(30) COLLATE pg\_catalog."default" NOT NULL,

receive\_notifications character varying(5) COLLATE pg\_catalog."default",

CONSTRAINT account\_pkey PRIMARY KEY (account\_id)

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public.account

OWNER to postgres;

**Association\_sponsors:**

CREATE TABLE public.association\_sponsors

(

asp\_id integer NOT NULL DEFAULT nextval('association\_sponsors\_asp\_id\_seq'::regclass),

association\_id bigint,

sponsor\_id bigint,

CONSTRAINT association\_sponsors\_pkey PRIMARY KEY (asp\_id),

CONSTRAINT association\_sponsors\_association\_id\_fkey FOREIGN KEY (association\_id)

REFERENCES public.associations (association\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT association\_sponsors\_sponsor\_id\_fkey FOREIGN KEY (sponsor\_id)

REFERENCES public.sponsors (sponsor\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public.association\_sponsors

OWNER to postgres;

**Associations:**

CREATE TABLE public.associations

(

association\_id integer NOT NULL DEFAULT nextval('associations\_associationid\_seq'::regclass),

association\_name character varying(60) COLLATE pg\_catalog."default" NOT NULL,

page\_link character varying(30) COLLATE pg\_catalog."default" NOT NULL,

initials character varying(10) COLLATE pg\_catalog."default" NOT NULL,

bio character varying(400) COLLATE pg\_catalog."default",

account\_id bigint NOT NULL,

location\_id bigint NOT NULL,

image\_id bigint NOT NULL,

CONSTRAINT associations\_pkey PRIMARY KEY (association\_id),

CONSTRAINT associations\_account\_id\_fkey FOREIGN KEY (account\_id)

REFERENCES public.account (account\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT associations\_location\_id\_fkey FOREIGN KEY (location\_id)

REFERENCES public.location (location\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public.associations

OWNER to postgres;

**Category:**

CREATE TABLE public.category

(

category\_id integer NOT NULL DEFAULT nextval('category\_category\_id\_seq'::regclass),

category\_name character varying(20) COLLATE pg\_catalog."default",

CONSTRAINT category\_pkey PRIMARY KEY (category\_id)

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public.category

OWNER to postgres;

**Events\_stats:**

CREATE TABLE public.event\_stats

(

stat\_id integer NOT NULL DEFAULT nextval('event\_stats\_stat\_id\_seq'::regclass),

stat\_date timestamp without time zone,

interested\_count integer,

event\_id bigint,

CONSTRAINT event\_stats\_pkey PRIMARY KEY (stat\_id),

CONSTRAINT event\_stats\_event\_id\_fkey FOREIGN KEY (event\_id)

REFERENCES public.events (event\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public.event\_stats

OWNER to postgres;

**Events:**

CREATE TABLE public.events

(

event\_id integer NOT NULL DEFAULT nextval('events\_event\_id\_seq'::regclass),

association\_id bigint,

event\_name character varying(100) COLLATE pg\_catalog."default" NOT NULL,

is\_live character varying(3) COLLATE pg\_catalog."default" NOT NULL,

category character varying(15) COLLATE pg\_catalog."default",

entrance\_fee character varying(10) COLLATE pg\_catalog."default",

location\_id bigint,

registration\_link character varying(200) COLLATE pg\_catalog."default",

description character varying(500) COLLATE pg\_catalog."default",

image\_id bigint,

start\_date character varying(20) COLLATE pg\_catalog."default",

end\_date character varying(20) COLLATE pg\_catalog."default",

end\_time character varying(20) COLLATE pg\_catalog."default",

start\_time character varying(20) COLLATE pg\_catalog."default",

time\_stamp timestamp without time zone,

CONSTRAINT events\_pkey PRIMARY KEY (event\_id),

CONSTRAINT events\_associationid\_fkey FOREIGN KEY (association\_id)

REFERENCES public.associations (association\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT events\_image\_id\_fkey FOREIGN KEY (image\_id)

REFERENCES public.images (image\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT events\_location\_id\_fkey FOREIGN KEY (location\_id)

REFERENCES public.location (location\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public.events

OWNER to postgres;

**Events\_categories:**

CREATE TABLE public.events\_categories

(

ec\_id integer NOT NULL DEFAULT nextval('events\_categories\_ec\_id\_seq'::regclass),

event\_id bigint,

category\_id bigint,

CONSTRAINT events\_categories\_pkey PRIMARY KEY (ec\_id),

CONSTRAINT events\_categories\_category\_id\_fkey FOREIGN KEY (category\_id)

REFERENCES public.category (category\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT events\_categories\_event\_id\_fkey FOREIGN KEY (event\_id)

REFERENCES public.events (event\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public.events\_categories

OWNER to postgres;

**Followed\_associations:**

CREATE TABLE public.followed\_associations

(

fe\_id integer NOT NULL DEFAULT nextval('followed\_associations\_fe\_id\_seq'::regclass),

association\_id bigint,

user\_id bigint,

CONSTRAINT followed\_associations\_pkey PRIMARY KEY (fe\_id),

CONSTRAINT followed\_associations\_association\_id\_fkey FOREIGN KEY (association\_id)

REFERENCES public.associations (association\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT followed\_associations\_user\_id\_fkey FOREIGN KEY (user\_id)

REFERENCES public.students (user\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public.followed\_associations

OWNER to postgres;

**Images:**

CREATE TABLE public.images

(

image\_id integer NOT NULL DEFAULT nextval('images\_image\_id\_seq'::regclass),

image\_name character varying(25) COLLATE pg\_catalog."default",

image\_path character varying(200) COLLATE pg\_catalog."default",

CONSTRAINT images\_pkey PRIMARY KEY (image\_id),

CONSTRAINT images\_image\_id\_fkey FOREIGN KEY (image\_id)

REFERENCES public.images (image\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public.images

OWNER to postgres;

**Interested:**

CREATE TABLE public.interested

(

event\_id bigint,

user\_id bigint,

interested\_id integer NOT NULL DEFAULT nextval('interested\_interested\_id\_seq'::regclass),

CONSTRAINT interested\_pkey PRIMARY KEY (interested\_id),

CONSTRAINT interested\_event\_id\_fkey FOREIGN KEY (event\_id)

REFERENCES public.events (event\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT interested\_user\_id\_fkey FOREIGN KEY (user\_id)

REFERENCES public.students (user\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public.interested

OWNER to postgres;

**Location:**

CREATE TABLE public.location

(

location\_id integer NOT NULL DEFAULT nextval('location\_locationid\_seq'::regclass),

room character varying(10) COLLATE pg\_catalog."default" NOT NULL,

building character varying(30) COLLATE pg\_catalog."default" NOT NULL,

city character varying(10) COLLATE pg\_catalog."default" NOT NULL,

CONSTRAINT location\_pkey PRIMARY KEY (location\_id)

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public.location

OWNER to postgres;

**Notifications:**

CREATE TABLE public.notifications

(

notification\_id integer NOT NULL DEFAULT nextval('notifications\_notification\_id\_seq'::regclass),

notification\_name character varying(25) COLLATE pg\_catalog."default",

date\_sent timestamp without time zone,

event\_id bigint,

notification\_text character varying(300) COLLATE pg\_catalog."default",

CONSTRAINT notifications\_pkey PRIMARY KEY (notification\_id),

CONSTRAINT notifications\_event\_id\_fkey FOREIGN KEY (event\_id)

REFERENCES public.events (event\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public.notifications

OWNER to postgres;

**Review:**

CREATE TABLE public.review

(

review\_id integer NOT NULL DEFAULT nextval('review\_review\_id\_seq'::regclass),

event\_id bigint NOT NULL,

user\_id bigint NOT NULL,

review character varying(200) COLLATE pg\_catalog."default" NOT NULL,

rating integer NOT NULL,

date\_created date,

CONSTRAINT review\_pkey PRIMARY KEY (review\_id),

CONSTRAINT review\_event\_id\_fkey FOREIGN KEY (event\_id)

REFERENCES public.events (event\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT review\_user\_id\_fkey FOREIGN KEY (user\_id)

REFERENCES public.students (user\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public.review

OWNER to postgres;

**Sponsors:**

CREATE TABLE public.sponsors

(

sponsor\_id integer NOT NULL DEFAULT nextval('sponsors\_sponsorid\_seq'::regclass),

sponsor\_name character varying(30) COLLATE pg\_catalog."default" NOT NULL,

page\_link character varying(30) COLLATE pg\_catalog."default" NOT NULL,

image\_id bigint,

CONSTRAINT sponsors\_pkey PRIMARY KEY (sponsor\_id),

CONSTRAINT sponsors\_image\_id\_fkey FOREIGN KEY (image\_id)

REFERENCES public.images (image\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public.sponsors

OWNER to postgres;

**Students:**

CREATE TABLE public.students

(

user\_id integer NOT NULL DEFAULT nextval('students\_user\_id\_seq'::regclass),

first\_name character varying(20) COLLATE pg\_catalog."default" NOT NULL,

last\_name character varying(20) COLLATE pg\_catalog."default" NOT NULL,

hometown character varying(25) COLLATE pg\_catalog."default" NOT NULL,

college character varying(50) COLLATE pg\_catalog."default" NOT NULL,

major character varying(25) COLLATE pg\_catalog."default" NOT NULL,

gender character varying(10) COLLATE pg\_catalog."default",

bio character varying(200) COLLATE pg\_catalog."default",

birthdate date,

account\_id bigint,

image\_id bigint,

CONSTRAINT students\_pkey PRIMARY KEY (user\_id),

CONSTRAINT students\_account\_id\_fkey FOREIGN KEY (account\_id)

REFERENCES public.account (account\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT students\_image\_id\_fkey FOREIGN KEY (image\_id)

REFERENCES public.images (image\_id) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public.students

OWNER to postgres;

**Transactions:**

CREATE TABLE public.transactions

(

transaction\_id integer NOT NULL DEFAULT nextval('transactions\_transaction\_id\_seq'::regclass),

type\_of\_transactions character varying(20) COLLATE pg\_catalog."default",

amount\_charged character varying(20) COLLATE pg\_catalog."default",

date\_done date,

CONSTRAINT transactions\_pkey PRIMARY KEY (transaction\_id)

)

WITH (

OIDS = FALSE

)

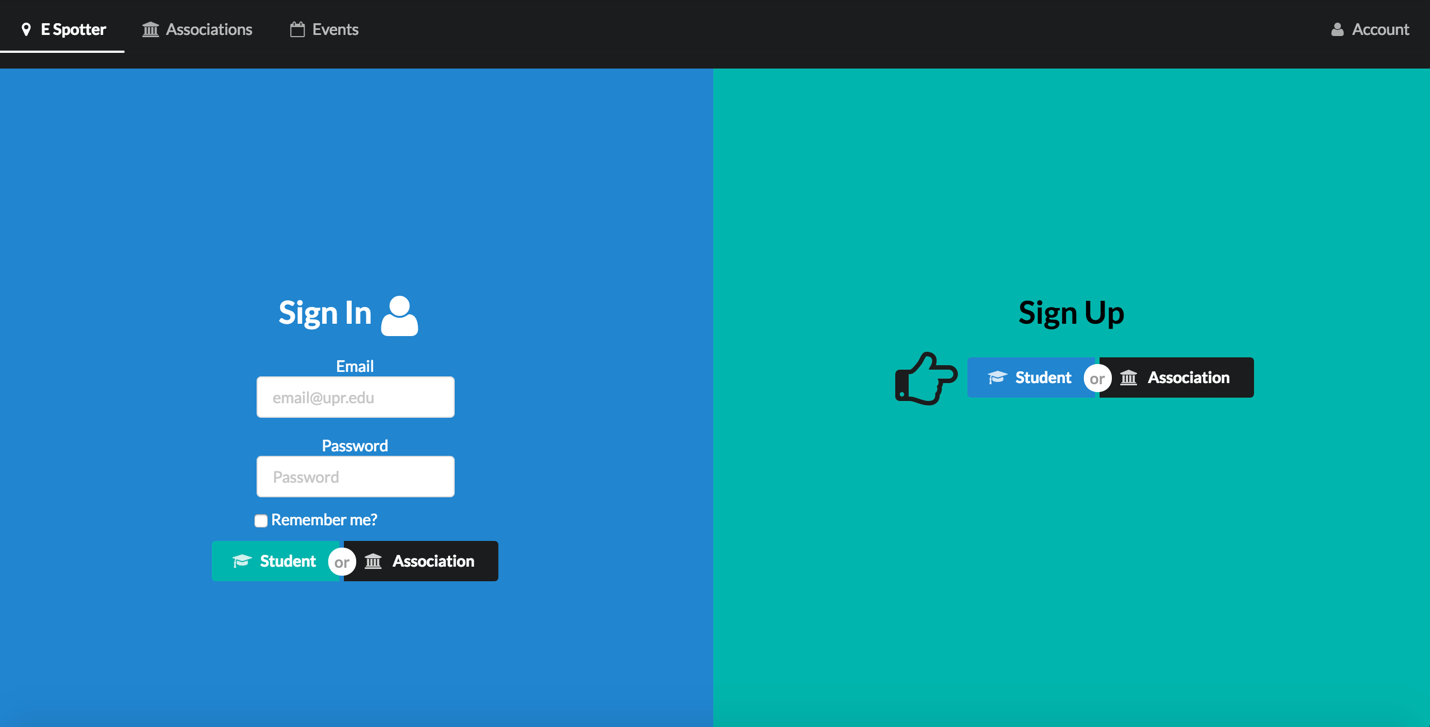
TABLESPACE pg\_default;

ALTER TABLE public.transactions

OWNER to postgres;

**5.** **Client – Side Screen Shots (refactored)**

**Login**



The new login for the web application in which the students and associations can sign up or sign in.

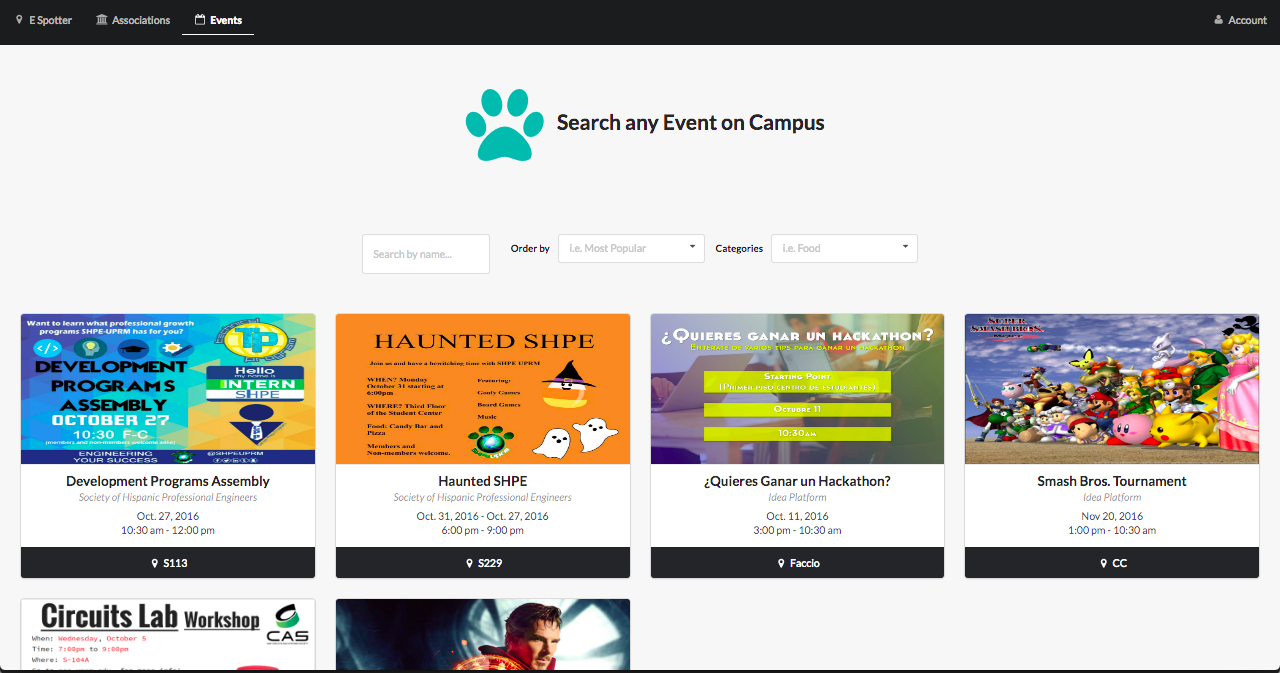
**Profile Page**



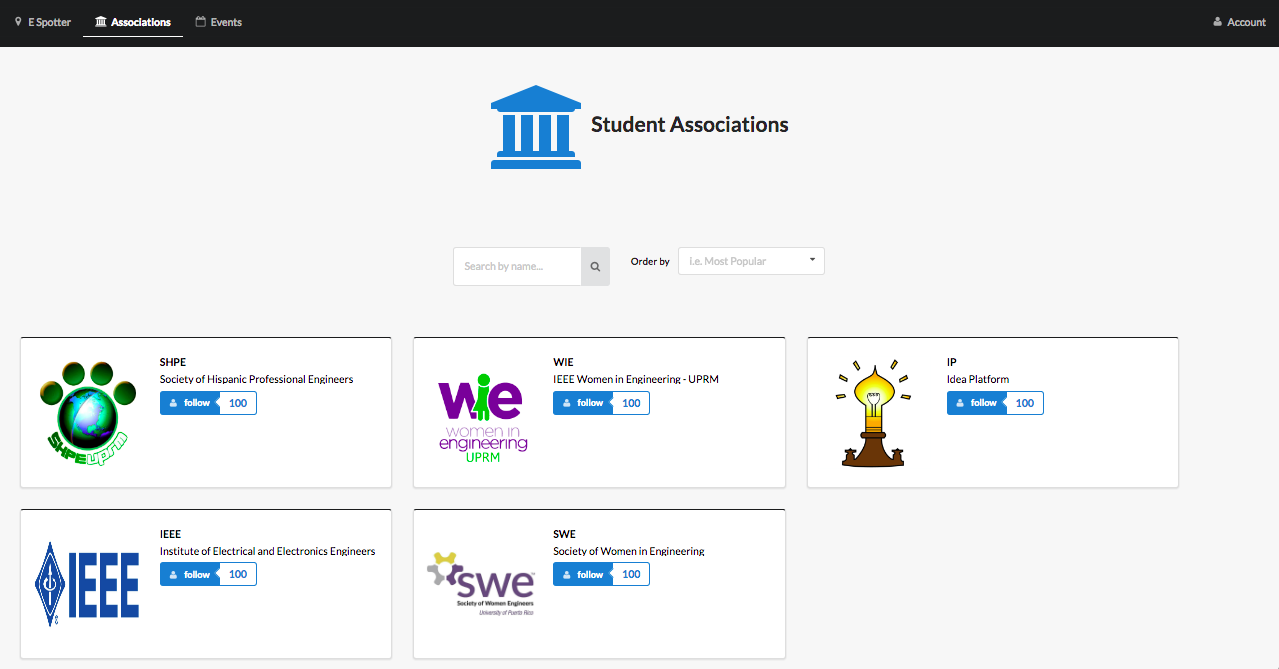
This page shows the structure for the student. It will include all personal information as well as the associations and events that he or she is currently following.

**All Events Page**

This page shows all of the available events that are created in our application. The user will be able to select any event and see all the the event’s information.

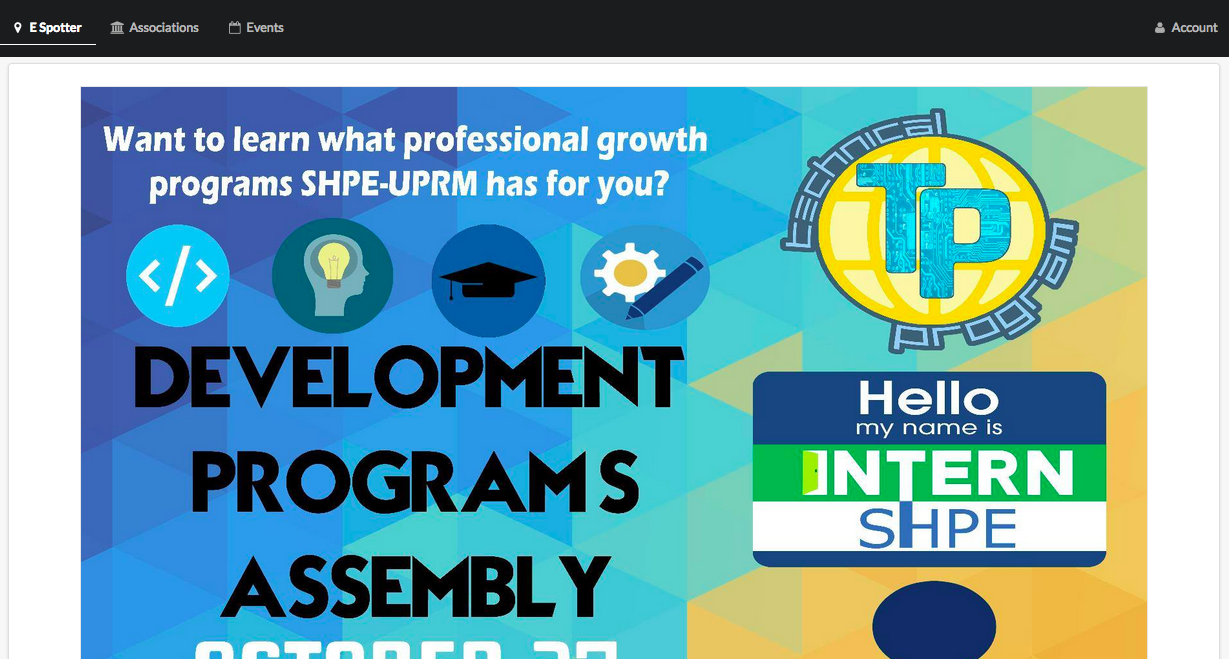


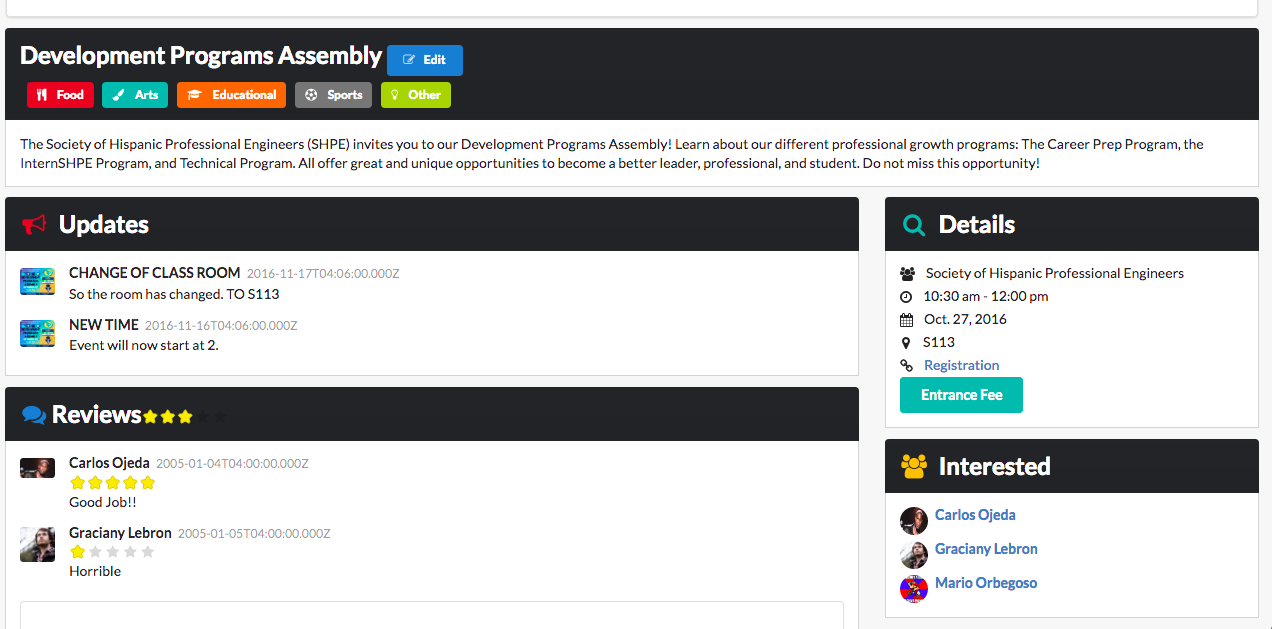
**All Associations List**



This page shows all of the available student associations that have profiles in our application. The user will be able to select any association and enter its profile.

**Individual Events**





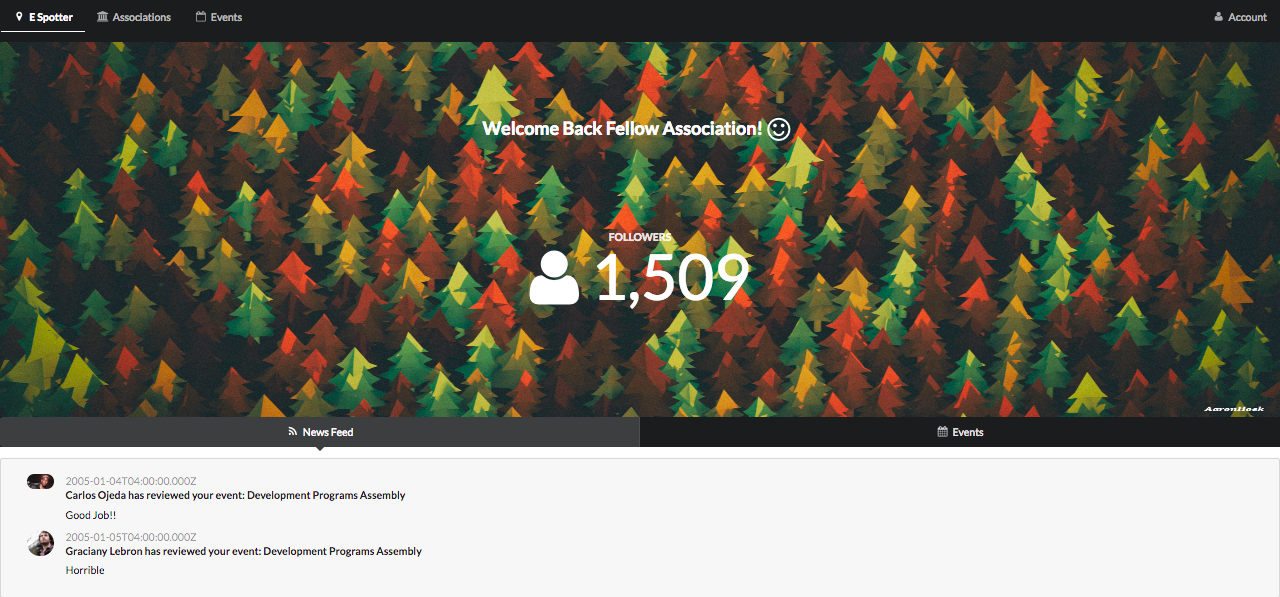
This page shows more or less the information for each event will be structured. It will also have reviews, updates, a list of the interested people and a graph showing the total interest.

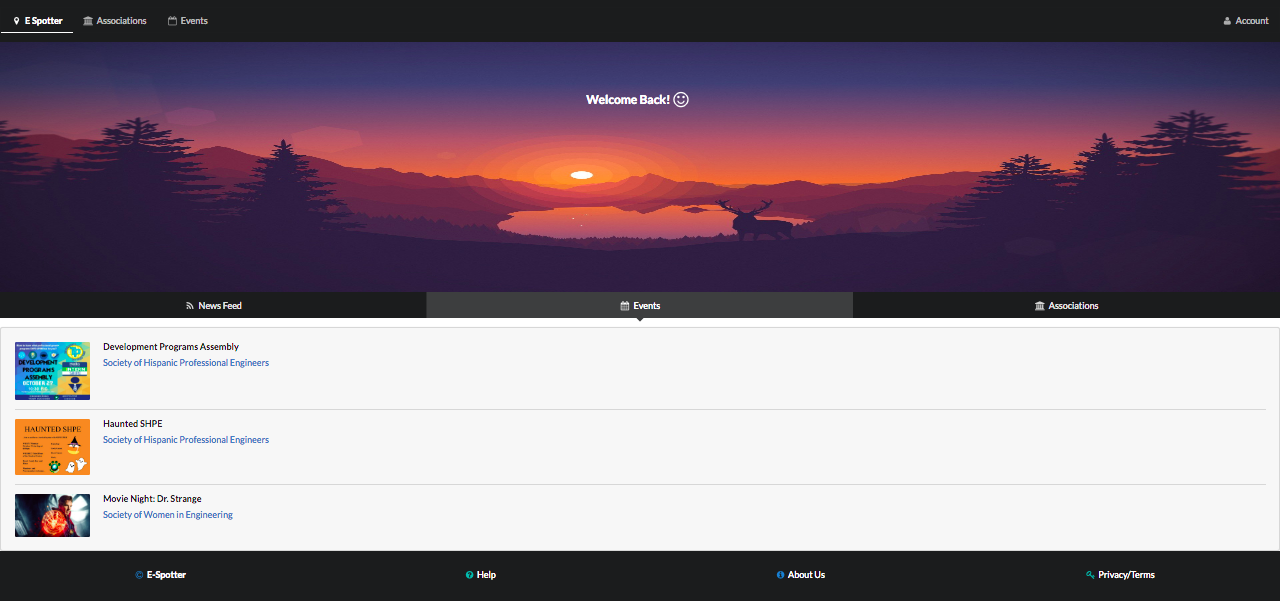
**Association Page**



This page shows how the information of a particular association. Any user will be able to see the events they are hosting or have hosted in the past, as well as a list of their sponsors.

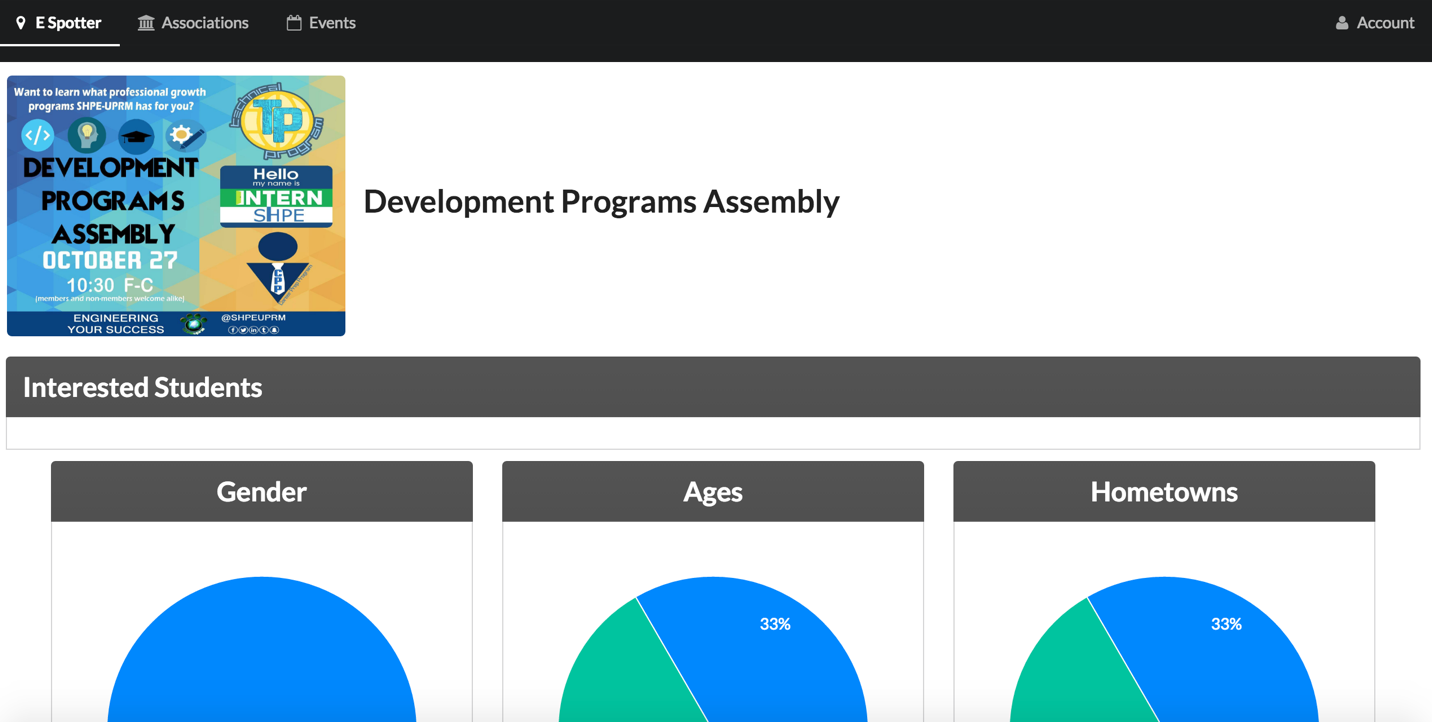
**Student Home and Association Home**

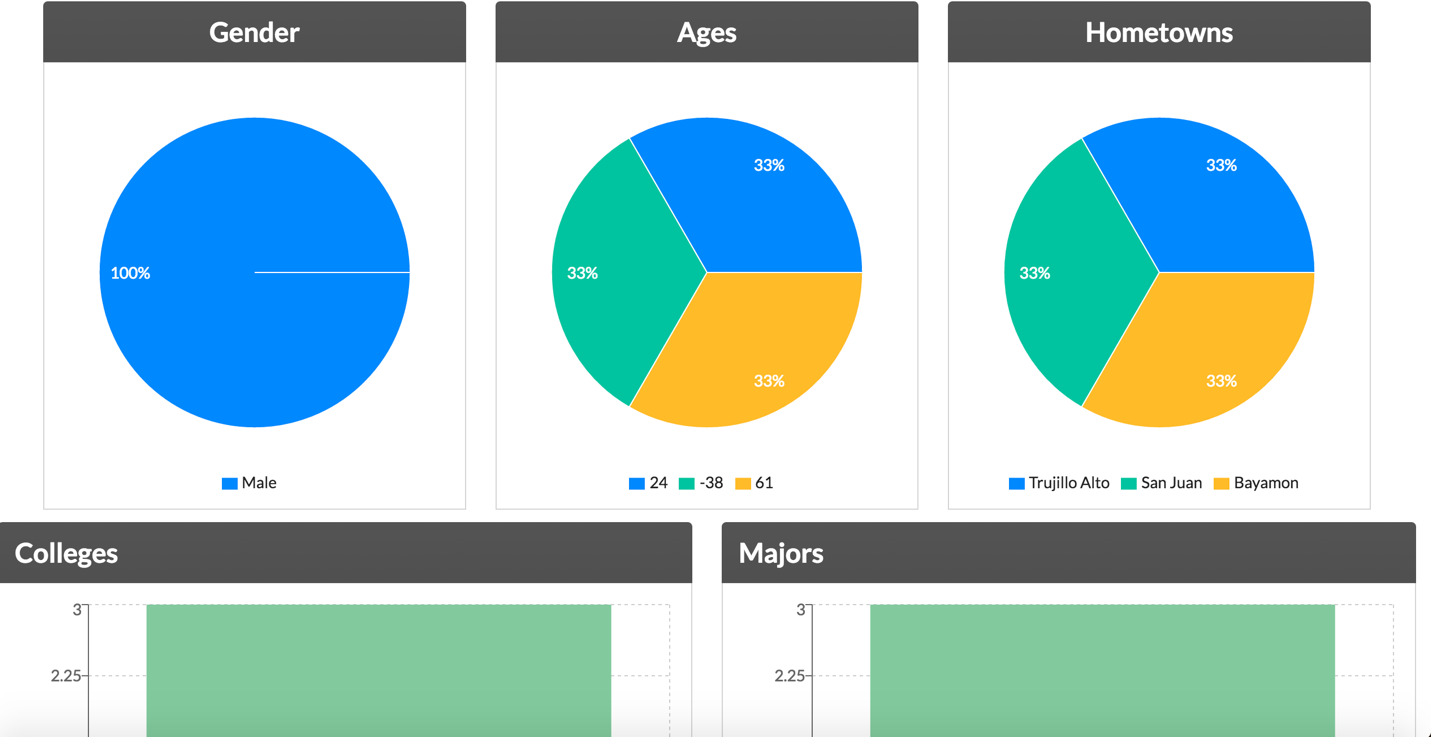
****

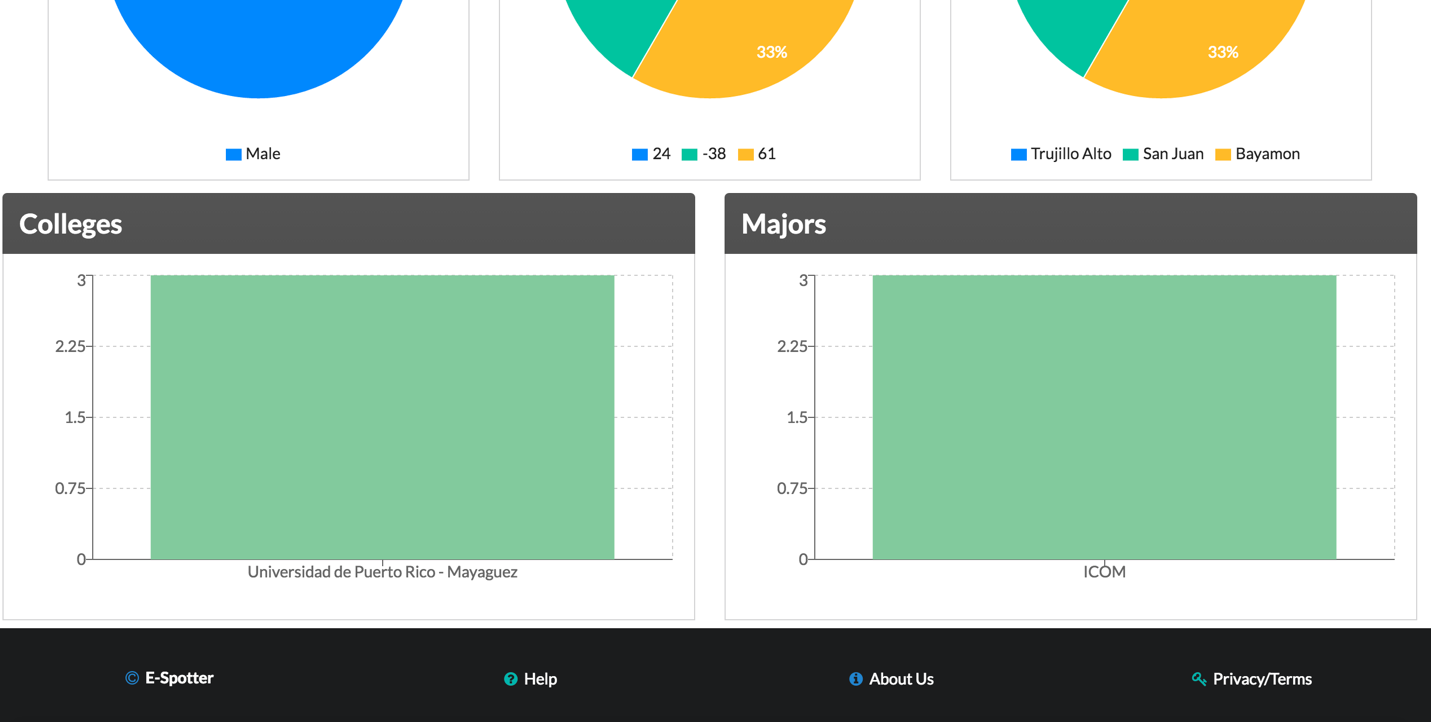
****

This view shows the home for the association and the student user. For the student they will be able to see notifications that the associations send, the events that they will attend and the associations that they follow. On the other side the associations will be able to see their events and some stats about them if they click a button, and they will also they see the amount if followers they currently have.

**Stats view for the associations**

****

****



These are some of the graphs and info the associations can see about their events so they can see which audience are interested in their events.