CSCI 3308 Final Project
EventGo

Developed by Jacob Ludwigson, Danny Alba, Michael Ellis, and Jacob Lancaster

Description

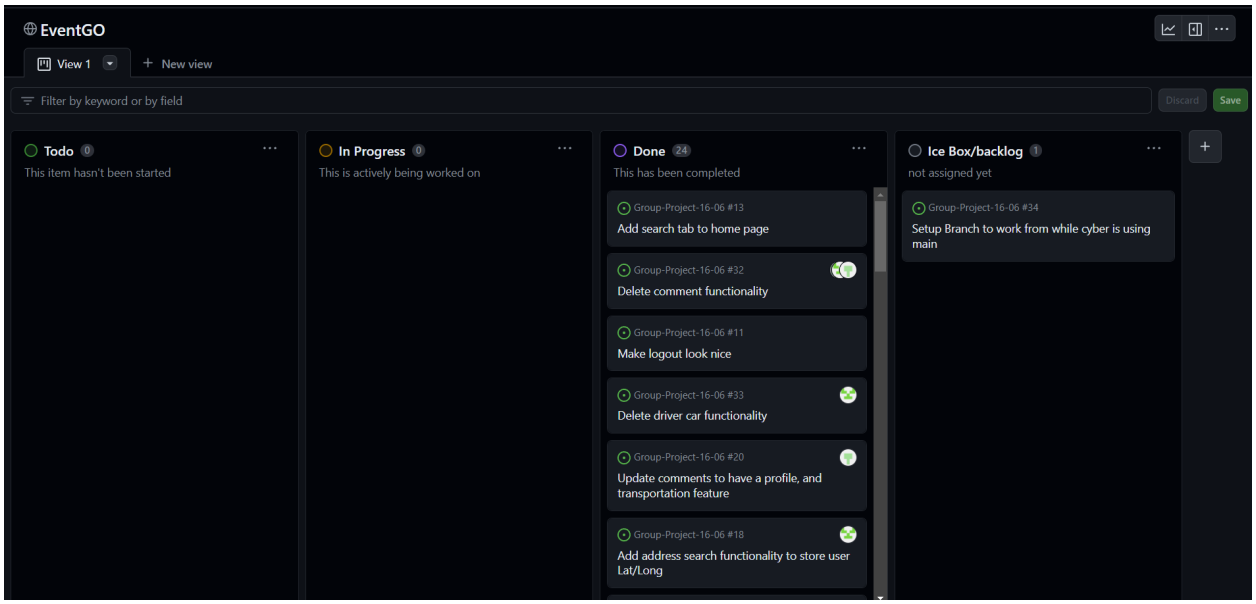
EventGo is a diverse application that is designed to bring communities together via events. What makes our application different from our competitors is the ability to find a ride to an event based on geolocation. Most applications that are used for events allow you to purchase tickets and find optimal seating. Our twist to this idea was to allow members of a community to comment and discuss an event of interest and then utilize further resources to arrive at the venue using our carpooling system.

Our team feels that there has been a gap between attending events and finding people to attend with. Building a stronger community means finding a way to allow others to connect, which brings us EventGo. With diverse capabilities pertaining to various physical properties, we hope that this application can bring you closer to people with mutual interests.

Our application utilizes data from SeatGeek and Mapbox external APIs which allows us to provide an experience where users can seek out transportation help or discuss event details. We think that providing this to the public will allow communities around the world to prosper. If you are interested in this product and its generalized ideas, please reach out to discuss any business related inquiries.

Project Tracker

[Link to Tracker]



Video Demonstration

[Link to video demo]

VCS

[Link to GitHub Repository]

Contributions

Jacob Ludwigson

Collaborative contributions on database (sql) and server routes (nodejs). Specific routes include initial design of /editProfile, /profile, /transportation, /driver,/removeComment, and /removeCar routes, although many of these routes evolved through collaborative efforts. I also contributed to the transportation.ejs,event.ejs,userprofile.ejs,driver.ejs and editprofile.ejs page logic as well as handled the Mapbox API axios calls + insertion in the pages.

Danny Alba

Collaborated mostly in the car ride and comment tables. Made design for event route and transportation route, contributed to UI in all pages, mostly in event pages where comments are handled, and rides can be shared. Worked in styling the event page mostly, the initial profile and transportation pages.

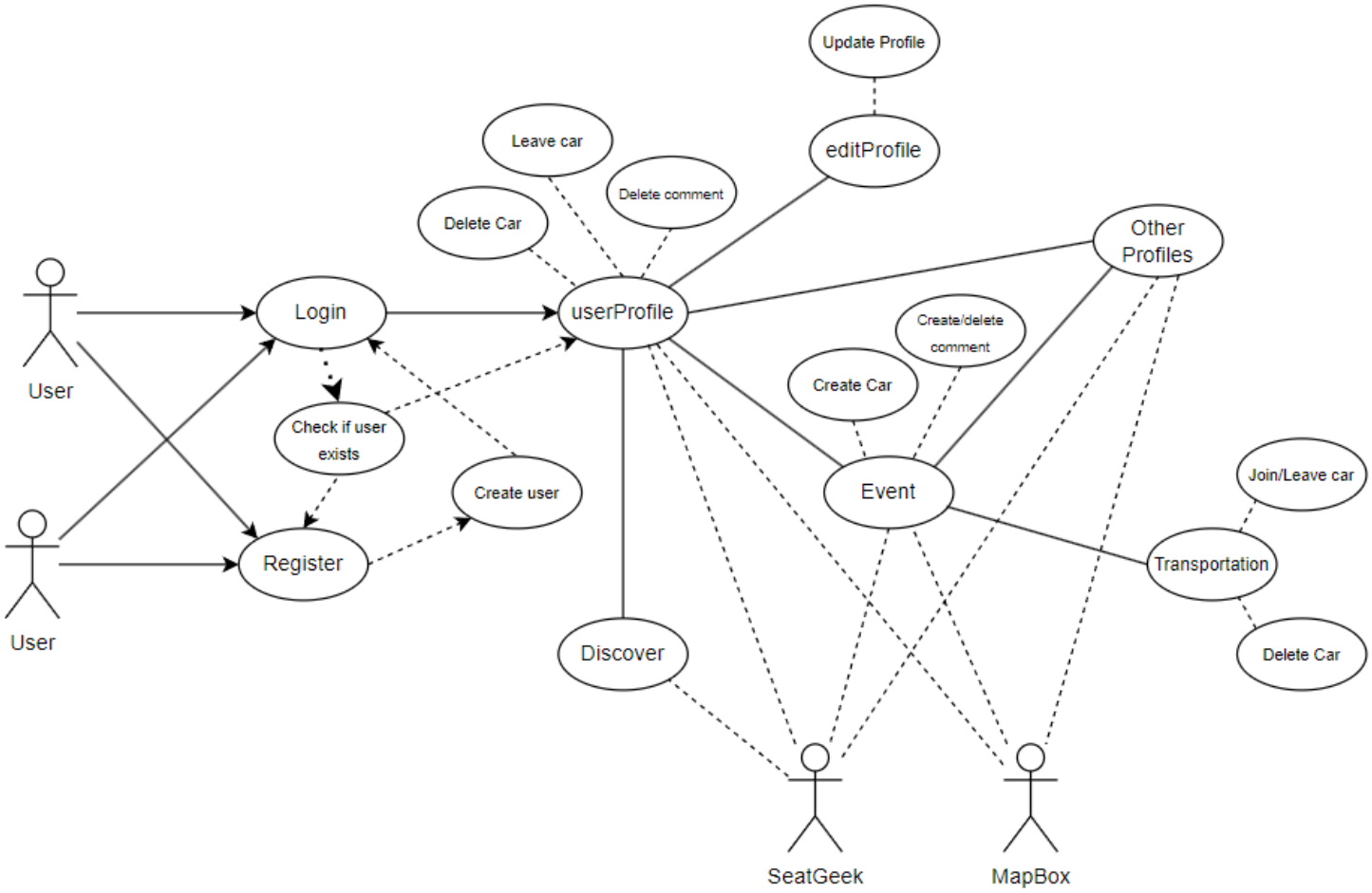
Michael Ellis

Worked on main functionality of the Discovery page; implementing the search bar, refreshing of the page, visuals for the page, and data imports from the SeatGeek external API. Also brought the SeatGeek API to our website. Contributed to styling of all pages and structuring of styles throughout Events.

Jacob Lancaster

I am responsible for the final styling and UI improvements to the profile, userProfile, editProfile, discover, transportation and event pages. I improved upon the \profile route by separating into two separate routes \userProfile and \profile. This change allowed me to populate the user's profile with information about the user's rides and comments and using an axios call to show event details and links specific to the user . Showing different information for other profiles. implementing the routes made by jake, \removecar and \removecomment on the userProfile page only. I made a similar improvement to \transportation by separating \leaveTranspoprtaion and \transportation to simplify the code and make it easier to read while ensuring the current user was only joining the event specific ride. I also made the use case diagram.

Use Case Diagram



Testing Results

Upon completion of Lab 11, our team decided to design a few more tests to determine the full range capabilities of our application. One of two important tests that were run was the refreshing and accuracy upon searching a new query on the Discovery page. We needed to ensure that new data was being presented both on the front and back ends of the application. Over time, we were able to successfully test and confirm the functionality of the discover page and its capabilities to dynamically update with user input.

The second important test was to ensure that when on a remote hosting server, users on different devices can see updates to changes in the Events and carpooling page. It took many failed testing attempts to discover root problems within our designed database. After fixing everything up, we were able to successfully see changes to the website when other users provided any inputs.

Some observations from other students outside our team that tested the features discussed in the UAT plan (login/discover/event discussion) include the addition of a more descriptive view on the discover page. The discover page searches events and displays the first artist at an event, so for something like a festival with multiple artists the discover page will load cards with only a portion of the event details/description because only the first artist is displayed. Observations regarding the comment section were overwhelmingly positive given the main functionality was achieved. The only additions that were suggested to the comment page by outside users were making the comments refresh without having to refresh the page, which would be achieved using sockets. Aside from suggestions, tests involving outside users were successful from a developer perspective because there was no unexpected behavior by the users that led to undefined behavior by our site. This was considered a success because our error handling kept the site from crashing and our functionality persisted throughout the tests.

Deployment

Remote Hosted Environment:
http://recitation-16-team-06.eastus.cloudapp.azure.com:3000/login

Local Host Environment:
To run this application on a local host environment the user needs to have docker installed on their pc, a SeatGeek API key, and a Mapbox access token. The user should clone the repository from the VCS (github), create a .env file as structured in the readME.md with the API key and access token. Once this is complete navigate to the ProjectCode folder in the repository and run the command “docker-compose up” to launch the container. Once tests run (if enabled by the user) and the terminal says “Listening on port 3000” and “Database connection successful” the user can navigate to their browser and in the search bar type “localhost:3000” to navigate to the login page.