Clippy

Gather

Fall 2017

Overview:

The purpose of Gather is to allow users and groups to quickly organize and post location-based events. Think SnapChat, for events! Users start by creating and posting an event to specific individuals or groups. Events have a name, location, and time, as well as an optional image and description with support for Google Maps, which when clicked on will open that location in a full Google Maps context. Users can add friends, create groups, and invite their friends to those groups. They can view upcoming events in various home, user, and group feeds and then either join or hide events. Viewing a group or another user's page will show the events shared and other users associated (member/friends) with that group or user. Users are able to edit their own account, groups they are a member of, and events they are hosting.

Team Members:

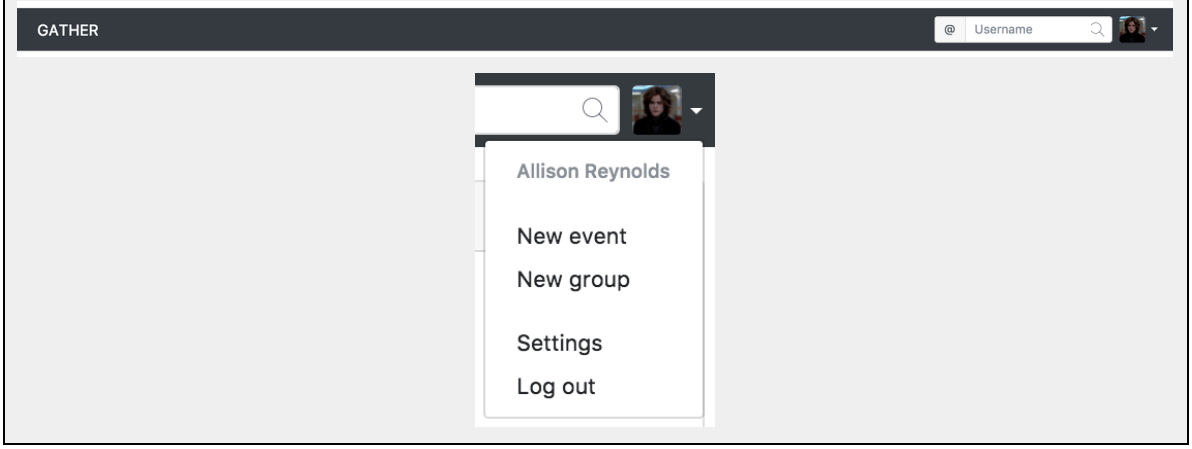
Luis Quiterio, Cyrus Freshman, Connor Ingram, Michael Maio, Ryan Clayton

Github Repository:

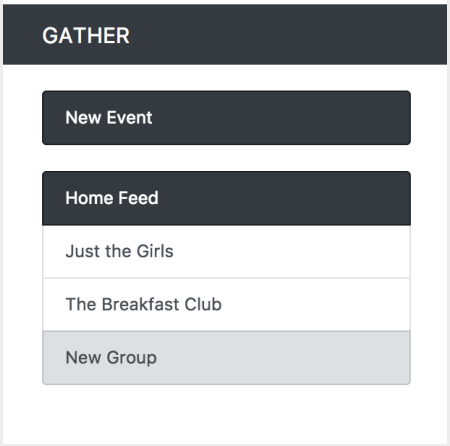
https://github.com/cfreshman/clippy

User Interface:

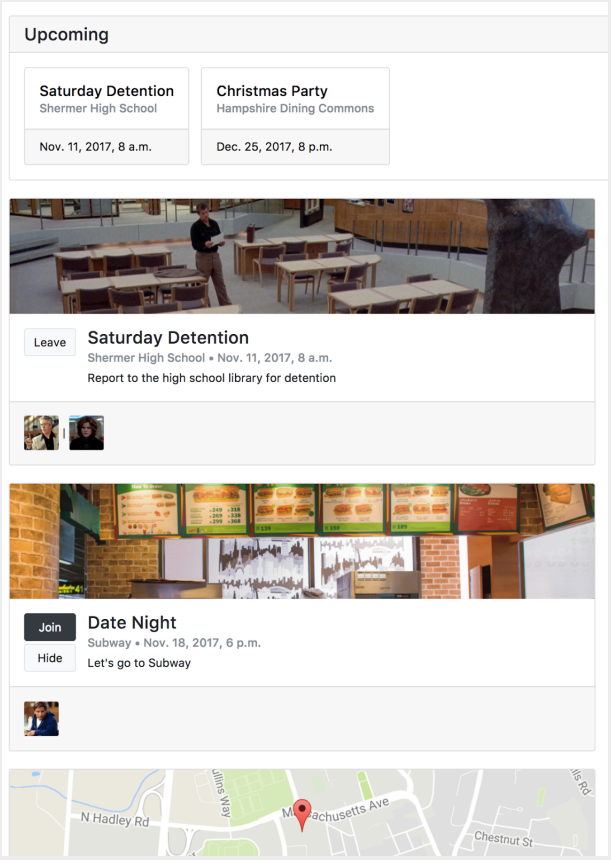
Navbar	Navigation, search by username, event/group creation, account options, log out



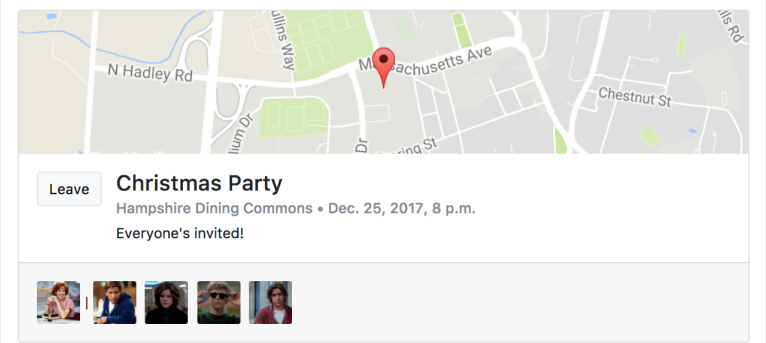
Sidebar	Quick navigation through user's groups, creation of new events or groups

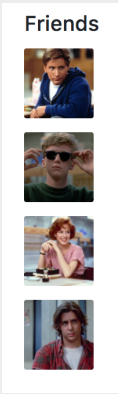


User feed	The user feed lists upcoming events the user has joined which can be clicked on to see more details of the event, and also a feed of events the user is invited to in the form of event cards.

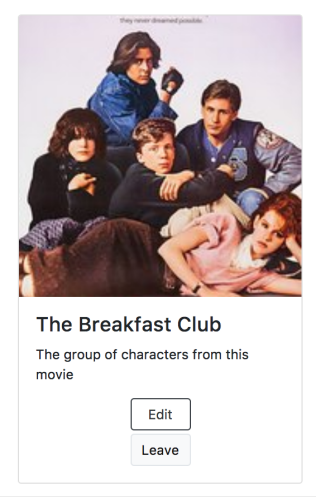


Event card	An event card displays information about a specific event. Each card includes an uploaded image or a Google Maps image of the location (which will open the location in Google Maps in a new tab), the event name, location, time, description, members hosting and members who

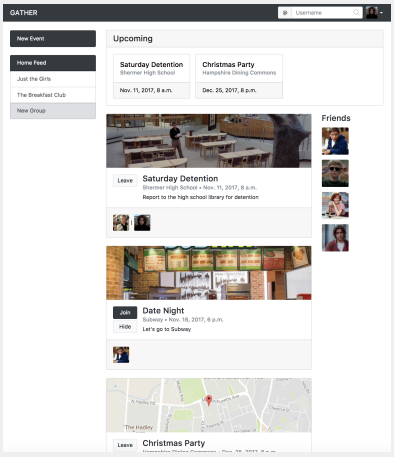
	have joined, and the option to join/hide/leave/edit the event. Hosts see ‘Edit’, joined see ‘Leave’, and invited see ‘Join’ and ‘Hide’.
	

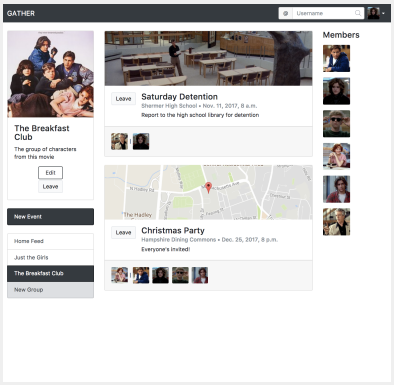
Friend List	A simple list of the viewer’s friends, displaying each profile picture. On group pages, this is the list of members, and on another user’s page, this is the list of their friends.
	

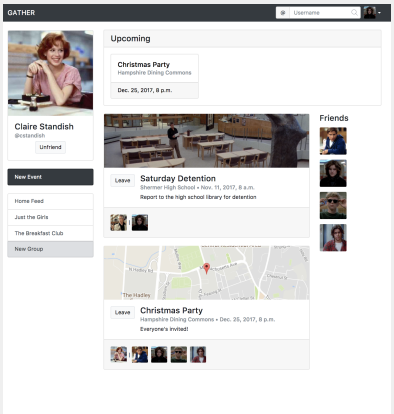
Group card	Shown on the group page, along with the list of members and event cards for the group's upcoming events. Allows members of a group to edit or leave the group.



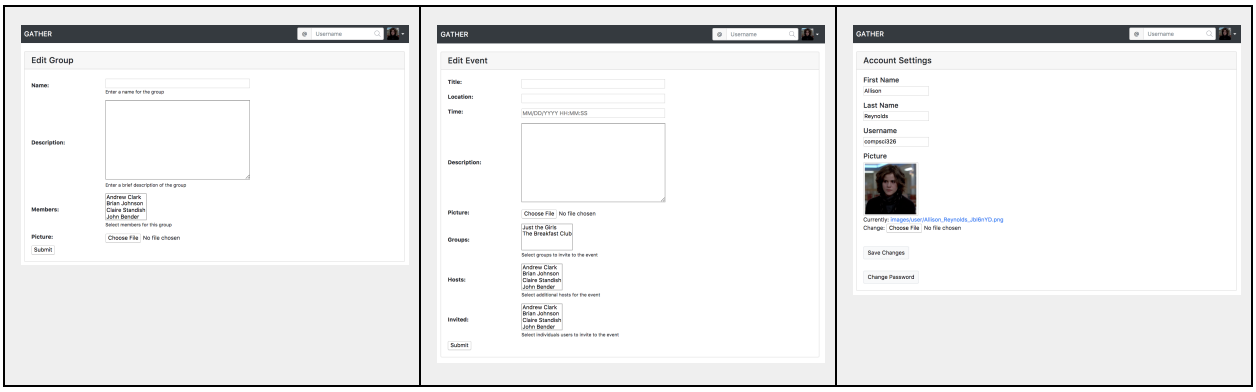
These above components come together to form three types of pages: the home feed, group feed, and user feed. The home feed includes all joined events in upcoming and all non-hidden invited events in the feed (events can only be hidden from this feed). The group feed includes only events posted to that group. The user feed includes mutually joined events in upcoming and mutually invited events in the feed.





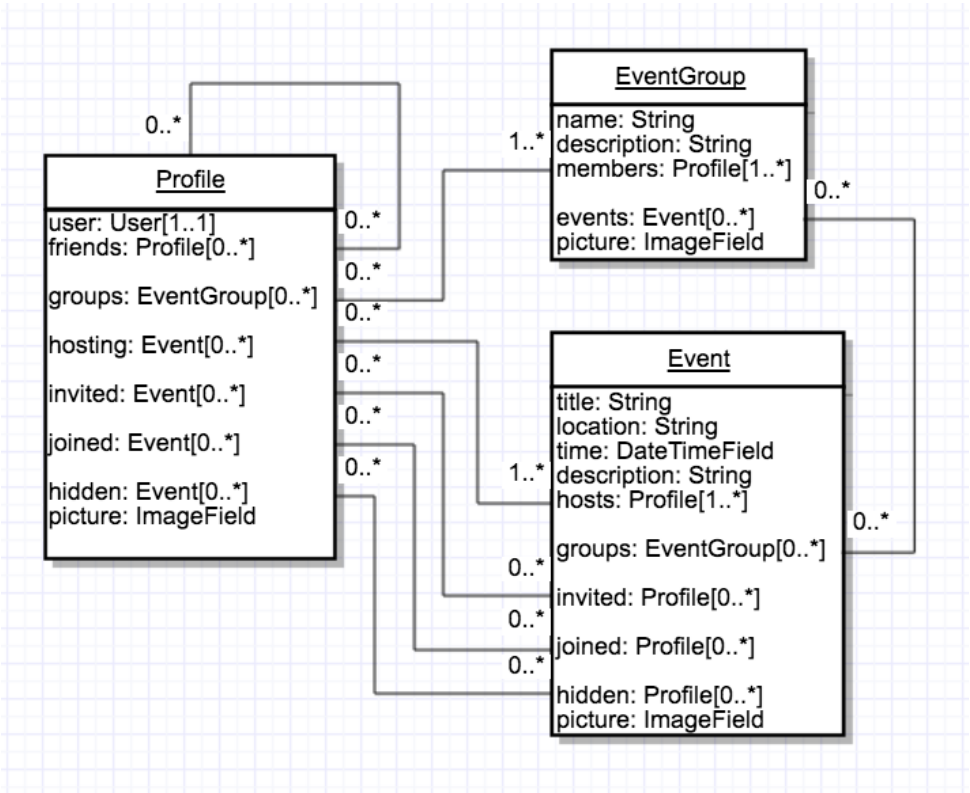


Additional views for creating/editing groups, creating/editing events, and editing account settings. The create/edit group view allows users to either create or edit a group by inputting information such as group name, description, list of members, and an image. Similarly, the create/edit event view allows users to create or edit events by inputting an event name, location, time, description, an image, and a list of users invited. The edit account view allows users to update their account information which includes their first/last name, username, and a profile picture.



Data Model:

Our data model has a Profile, EventGroup, and Event. The profile is attached to a User and will store the data for each person's profile. The EventGroup model contains users who are members of the group. This model exists to provide easy grouping for friends so they can send 1 event invitation instead of many. The event model is the core of our application, this stores data for the users about created events that users can be invited to. Each profile has one user model, friends which are other users and a picture. Event groups have a title, description, user members and a picture. Events have a title, location, time, description, a picture, then multiple areas for members to either be invited as a group, individually, joined, or to hide the event from the user.



URL Routes/Mappings:

/accounts/login	Login page – user will be redirected here from any page if not logged in
/	Reroutes to /gather/
/gather/	Index
/gather/g/new	Create group
/gather/g/(?P<id>\d+)	View group
/gather/g/(?P<pk>\d+)/edit	Edit group
/gather/e/new	Create event
/gather/e/(?P<id>\d+)	View event
/gather/e/(?P<pk>\d+)/edit	Edit event
/gather/u/(?P<id>\d+)	View user
/gather/u/edit	User settings
/gather/u/edit/password	Change user password
/gather/u/(?P<id>\d+)/(?P<action>\w+)	URL map for a user action (friending/unfriending)
/gather/g/(?P<id>\d+)/(?P<action>\w+)	URL map for interacting with a group (leaving/joining)
/gather/e/(?P<id>\d+)/(?P<action>\w+)	URL map for interacting with an event (join/leave/hide)
/gather/search/	Used for finding for user profiles

Authentication/Authorization:

Users are forced to login in order to access the application, since the content on every page depends on the current viewer—the viewer will only see events they are invited to (or hosting). They will be redirected to the login screen if they aren't already logged in or if they proceed to logout. All users have the same permissions throughout the website. Django's form of permissions doesn't work for this model, but we have defined many ways the view is catered to the current user based on what they should be allowed to do. Any user that is a host of an event has the ability to edit that event, whereas others that are just simply invited may not edit it. Accordingly, only the host(s) of the event will see the edit button available. Only members of a group are able to view that group's page.

Team Choice:

Our team choice was to implement the Google Static Maps API into our application, being used as a feature in events. If an image is not uploaded for an event, a static maps image will be generated based on the event location for the top of the event card. Clicking on the image will open the location in a new Google Maps window for the full context and directions. This required getting a Google Maps API key and enabling the Google Static Maps API for that key. To request the map image, we encode the event location as part of a valid URI query that generates a map with the desired properties. Even if the user uploaded an image (and so the map is not shown), we link the image to a Google Maps search of the location. Our group also implemented image uploads and heavily integrated the uploaded images into our application, which was not covered in class.

Conclusion:

Our team learned how to work in a group environment towards a common goal. It was an interesting and informative experience to work through the very iterative process of how an application comes together, starting with brainstorming, mockups, and designing a data model, and finally by the end adding finishing polishes and last-minute touch-ups. We all agree it would have been nice to be more experienced with git version control before this project. However, the greatest difficulty we encountered was with the conflict between our models and Django's authentication. If we had known about Django's User and Group models before creating our model, it would have been a much smoother process to plan and integrate our own models into that existing system, rather than trying to patch them together and avoid conflicts afterwords.