CS316 Open Project
Sam Ginsburg, Ethan Gottlieb, Joe Jacob, Greg McKeon
Final Project Report
12/9/15

Spawn

Project Description:

Spawn is a social networking web application that allows you to join or “spawn” events to enjoy spontaneous meetups. Whether you’re in the mood to get a quick bite, play pickup soccer, see that new movie, go on a hike, or head to a party, Spawn brings the right people together at the right time.

Glance through upcoming events (all within 24 hours) created by your Facebook friends or by nearby users and view the event type, distance, time, attendees, etc. If an event strikes you, join and connect with other attendees! If you can’t find what you’re looking for, “spawn” an event of your own! In the process, you’ll catch up with friends you don’t see often and meet fun people.

We have created a fully-functioning web application that allows users to login and create or join events. The UI and UX are simple, intuitive, and reactive, only including information that is important to users. The main page of the web application consists of a list of nearby events, and users can query and sort events by certain attributes (type, time, location, etc.) to help them find the perfect event. Users have their own profile pages and our connected automatically with their facebook friends.

This application is useful and interesting because there currently are no other websites/apps that encourage spontaneous meetups. Most other event apps are about helping to plan future events, while Spawn stresses the here and now. We could all imagine using this event to hang out with new people every day.

Technical Description:

We used the Meteor JavaScript framework to build the web app, in addition to HTML and CSS. To build the UI of our web app, we used the Bootstrap package for Meteor. Meteor uses MongoDB, which we used to store event and user data, and has features that help with processes like logging in.

Although the possibilities of events are endless, they all have most of the same attributes including title, description, maximum participants, visibility (public or private), time, and location. Each event also has a user that is the creator (with additional event host options), and a list of users that are attendees. Users also have similar attributes

that we populate mostly using Facebook, including relationships between users based on connections like location and “friends.” All the data in Spawn is user generated.

The code structure follows standard Meteor guidelines. The application directory is split into three main directories: client, lib, and server. The lib folder contains all collections, the app router, and global variables and functions. In Meteor, the lib directory is compiled first because many of its elements are essential to many features of any application. The server folder contains server code including the setup of Facebook connections. This is fundamental to the application because a user cannot add or join events without logging into Facebook. The client folder contains all CSS, stored in the stylesheets subdirectory, and HTML for the app is in the form of templates along with its companion JavaScript code, stored in the templates subdirectory. The templates subdirectory contains many different templates that contain code for individual UI and backend components of the application. This allows us to reuse templates for different aspects of the application without needing to rewrite code.

Meteor Technical Database Details:

The Meteor.js framework was built around making sure that apps are able to effectively and instantly update in real time based on changes to data in the database, as well as making sure that the front-end UI is still perfectly responsive and not “laggy” or slow due to needing to always wait for responses from the server. This led to some interesting implementation details worth briefly describing.

The directories in a Meteor project correspond to which side of the application will run the code within them. For example, code in the “client” or “server” folders will only be visible to the application clients or server, respectively, but code in the “lib” folder can be seen and used by both, and this folder is used for declaring the MongoDB collections used by the app. While a client is running a Meteor app, the server owns a complete copy of all the app’s data and user information, but the client also has its own, locally cached, smaller subset of every DB collection. The server selectively “publishes” these data subsets to clients which “subscribe” to that data as necessary based on need, security, privacy, etc. When users interact with the app’s UI in any way, the client instantly behaves and responds using its locally cached data so that the interaction is fast and smooth. The fact that the client’s data is only a small subset keeps the performance fast even when the size of the DB on the server becomes large. If the user’s action also requires interacting with the server in any way, the client “predicts” the results as best it can based on its local data and responds in that way to maintain the illusion of instant response. When the actual server response completes, any corrections are made quietly and quickly in a way which the user likely never notices.

To allow for the instant data updates effect, whenever a client changes local data which is then synced up to the server, the server forcefully sends an “invalidation” signal for that data to all other clients which have dependencies on that data. Then, when clients receive this signal, they identify all page elements which were tracked to depend on those invalidated collections and silently update only the page elements and their locally cached copy of the DB collection with the newly received data from the server without needing to refresh the whole page.

For much more detailed information about the Meteor javascript framework, visit https://www.meteor.com/.

Database Description:

We use MongoDB for our database. MongoDB is a NoSQL document-based database which primarily stores information as collections of JSON data. Our collections include:

1) Events, with a primary key Event_ID
2) PastEvents, with a primary key Event_ID
3) User, with a primary key User_ID
4) EventComments, with a primary key Event_ID

Their schema are defined as:

Event(id, Title, Description, Owner, attendees, location, date, duration, maxPeople, private)

ID is a unique number representing the actual event. This is the primary key for the database, since it is the only property guaranteed to be unique. Title is a string which represents the name of the event, while Description is a String that contains the details of the event, as set by the owner. Attendees is a list of id’s of Users who will be attending the event. Location is a pair of floats that represent the coordinates of the location (or, for MVP1, a String with the address). Date is a date object representing the start time of the event. Duration is an int representing the event length in minutes. MaxPeople is an int containing the max number of people who can attend, while private is a boolean indicating whether or not the event is publically visible.

User(User_ID, FB_ID, name, friends)

The friends attribute for a user is only updated (loaded from Facebook) at the beginning of each session (login) and when that user’s profile page is viewed. This is to

avoid unnecessary repeated requests to the Facebook Graph API to retrieve the friends of a User, and will allow for faster access to events made by friends during the session and will be updated with new/deleted friends at the start of each session or as they are viewed. This allows us to query a local, minimal copy of our database, which is much faster than waiting for API data.

For now, we do not store any settings information in the User table. As we develop the project, particularly the profile piece, we will use this table to store User preferences and settings, as well as profile information we cannot pull from the Facebook login.

As of now, we've decided to allow Users make events that occur at the same time and location because multiple Users that are unrelated to each other should be able to make an event at the same location given that the location is big enough; however, we'll consider adding a notification that will alert a User that another event is occurring at the same location and time as that User's event.

PastEvent(id, Title, Description, Owner, attendees, location, date, duration, maxPeople, private)

The "PastEvent" collection has the same schema and keys as the "Event" collection. It is used to save events that have already happened in the past and were removed from "Event" for being expired. This is used as history so that each user's profile page can be populated with a list of their event history. Once events are moved into "PastEvent", they are no longer able to be modified by app users.

EventComments(Event_ID, author_name, message)

This database collection is used to store users' comments which are each associated with a specific event. Whenever a user submits a comment for an event, their name, the event's ID, and their message are recorded. This collection is kept separate from the "Event" collection rather than simply being a field within it.

This separation was done for a couple reasons. First, users likely will add comments frequently while the event's data remains mostly constant, so it would be wasteful to need to completely send the entire event's data from the server each time an event is posted and the page is reactively updated, so separating this "EventComments" collection allows it to be updated and retrieved independently. Also, separating the comments allows for fine-grained control of how many comments to retrieve from the server and store locally on the client in the case that the number of comments on an event becomes large. If not for this separation, the client would be

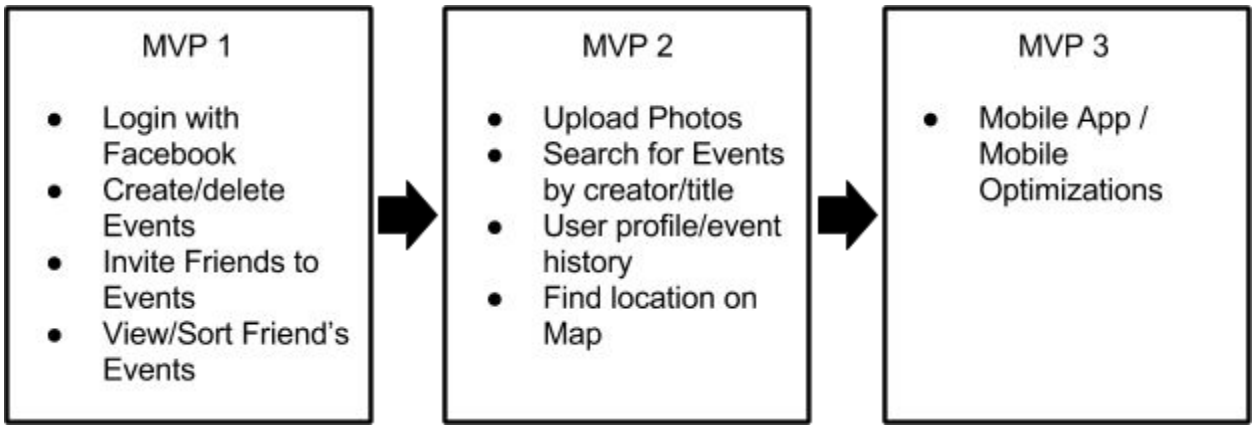
forced to load and cache every comment submitted for that event whenever retrieving an event's data or updating the page for newly submitted comments.

Searching and Sorting:

Searching and sorting are incremental in this application because on the main page, all events are constantly populated through a search. This is accomplished with the easy:search package, which allows users to search MongoDB collections given a search parameter and criteria.

Search criteria (sort by, direction of ordering, filter by) are stored in a local reactive variable that updates whenever a user changes a search choice. Any update to this reactive variable triggers an automatic new search to display the results in the desired format.

Summary of Final Progress:



We were able to complete most stretch goals and produce a production-ready web app. These stretch goals include incorporating maps, commenting, social features, and enhancing UI. The app is fully-functional, and we were able to improve the UX and flow throughout, as well as fixing bugs. Our app even runs on mobile, although the UI has only been optimized currently for larger computer screens.

The below list includes the major features that we have completed:

- Facebook login and integration
    - Pulled user information for profile pictures, gender, age, and friends
    - Splash page displayed if user not logged in
- Reactive main page displays nearby events

    - Each element in the list shows preview details including name, host, distance, number of attendees, and countdown timer
    - Click on event to be brought to event page
    - Events joined displayed in tile view at top of page
- Event searching and sorting
    - On the home page, users can sort by time to event, distance, event name, and others
    - Users can also search by name, host, and description
    - Users can filter for public vs private, to see only their friend's events vs all nearby events
- Event creation page
    - Have their own description, name, time, capacity, private/public, location
    - Unique identifier is created for each event
- Event detail page
    - Event attributes of name, host, time, and description are pulled from the MongoDB database and shown on the page
    - Map shows your location and event location
    - All attendees are shown
    - Comments on events are updated in real time and displayed with user name and time posted
    - Users can join an event to follow it and display intent of attending
    - Users can also drop events, and event hosts can delete events
- User profile page
    - Contains profile picture pulled from Facebook along with basic information
    - Shows past events that User spawned
    - Users can query other Users for their profile pages
    - Can proceed to user's facebook page, and view all their friends using the app
- Archive events once completed
    - After an event starts we archive the event to a pastEvents collection
    - pastEvents are displayed on a user's profile page and includes events he/she attended and hosted
- Google Maps API for location support
    - Each event has a location associated with it, which defaults to the host's current location
    - We only need to store latitude and longitude for each event location
    - Hosts can search for places as well (restaurants, parks, etc.), and the location name will be included
    - The location's formatted address is displayed

    - Users can view their current location, the event location, and their calculated distance to the event
    - Users can click on the distance to an event to be redirected to Google Maps directions to the event, based on their current location.
- UI
    - Our UI has been significantly redesigned, making extensive use of reactive elements with Bootstrap

Summary of Open Issues and Future Work:

Our app is fully-functional, and we have completed all the necessary database work for releasing the app to the public. The next steps are improving UI and user-testing. Specifically, we would like to better optimize the UI for use on mobile devices as well. We will likely add new features in the future, including inviting specific friends to events, adding the ability to allow hosts to edit their events, uploading media such as photos to events, and allowing users to update profile settings such as blocking users, event notifications (email, texts, or sounds), and more.

Deployment:

Briefly, we used an Amazon Web Services VM to host our application, and we set it up by using a program called “Meteor Up” (mup) which mostly automates the process of deploying a Meteor application into production. For more information and instructions on how we deployed Spawn or how to do it yourself, see the README file included in our project and on Github, or read more about how to user Meteor and Meteor Up with AWS.

GitHub: https://github.com/sam-ginsburg/Spawn

Application Website: http://ec2-52-90-146-92.compute-1.amazonaws.com/