Kanav Dhir
Deven Dayal
Alexander Wong

Buzzy - Ruby On Rails Project

Background - About Rails

The Buzzy app was created using what all of us had learned about Ruby on Rails so far. In planning our app, we thought about a few key things:

- MVC: the Model-View-Controller architecture is the heart of Ruby on Rails, and creates a simple flow for passing information from databases (models) to views (basically what the user sees) through logic controlled by the controllers and routing. Our knowledge of this subject allowed us to easily diagram out our ideas in a simple way, and helped us to think about how we could improve different pieces of our app once we actually started coding.

- Routing: routing was one of our main challenges. Routing is tells Rails which controllers and methods to use to handle different URLs and HTTP requests. Routing involves the idea of a RESTful architecture, for example using GET, POST, and CREATE methods to control the actions of the application. This was a new concept for all of us at the beginning of the year, and working on this project helped us to learn a lot more and become comfortable with it.

- Test-driven Development: test-driven development is another key concept of Rails. While we did not use many test scripts because it is a somewhat advanced topic, it is basically just coding a feature to fail on purpose and letting Rails tell you what to fix. The notion of test-driven development was still very important and allowed us to develop separate pieces of the app very efficiently. In the future we hope to learn more about using testing methods such as RSpec.

- Databases: although using databases and models with Rails is fairly easy to do without having to understand much about database management (e.g. MySQL), we did have to learn a bit about CRUD operations and the basics about how database migrations work. One of the best part of Rails is that it manages a lot of this very easily.

- Views/HTML/ERB: Views are the surface of our web application. Every controller corresponds to a view, which is essentially HTML/CSS for the design of the specific page. In our views we were able to take advantage of ERB, embedded ruby. Embedded ruby is a templating system that embeds Ruby into text document, in our case, into our HTML. This gives us access to Ruby scripts from right within our HTML file.

Idea

The idea behind Buzzy was to create an app similar to the popular Waze app, which used crowdsourced data to provide accurate traffic information. We wanted to use a similar voting system which would allow students to see how busy different places on campus were, be it the gym, the library, or a certain dining hall. In addition, we wanted our app to be both web and mobile so that students could easily vote on how busy a place was and be able to see the busyness of a place conveniently.

Overall Structure

Our app consists of 3 basic classes/models: users, places, and votes. The structure is fairly simple, with a user being able to cast votes for different places.

A user is attributed, at the moment, with only a username and a password and the ability to create votes. Eventually, we see potential to incorporate gamification and to allow, for example, users to have scores and earn rewards for contributing to our app by voting.

Each vote corresponds to a certain place, and includes a numerical score (0=empty, 50=normal, 100=busy) as well as a timestamp. The timestamp attached to each vote means we can look at the busyness of a place over different periods of time.

A place has a name, a GPS location (something we have not really included in our app as of yet), and a current busyness score. This score is recalculated when a user views a place's page and can be calculated for different ranges of time.

Details

Alex focused on the "Users" model, which dealt mainly with being able to have unique users on the app in which you could log in and out and have a personal profile. To implement this, we essentially had to create first an entry in the database called Users, for which name, email, date created, and a password hash were all stored. To be able to store and use this information, we had to create both new user forms and sign in forms. These forms had to do some complex things - it had to make sure that for new users, a used email could not be used again, an email was in the proper format, passwords were a certain length, and names were in the right format.

One of the main points of having Users was to control what certain users could do, so we implemented a permission system in which only logged in users could see certain pages. This meant we had to make a session manager - when you logged in, a session was created just like on every other website that allows logging in/out. For ease of testing, the requirement of logging in to perform certain actions is currently disabled, but it will be enabled in a production version of Buzzy.

With the user model also comes a user profile page. Currently, it uses a Gravatar for a profile image and does not display any other information, but there are many uses for it in the future.

Kanav focused on the “Votes” Model, building the object created every time a user casts a quiet/moderate/busy vote. We decided to make a vote its own model/object rather than just an attribute of place or user so that eventually we could use data from each vote to understand more about each user and each place. We also wanted to eventually add some gamification component to our app which would also act as an incentive for users to provide information about places.

Each vote had major attributes, for example each vote belonged to both a specific user as well as a specific place. Each vote was also paired with a timestamp which, when each vote is accessed by the places, is used in our algorithm to help determine the busyness of the place. Right now the user must visit the “Vote” page to cast a vote. When the user visits this page, he/she chooses a location to which they want to direct the vote and then three buttons, quite, moderate, busy, to cast their vote. Each button is linked with the variable “score,” which is also used by our algorithm, and each button submits a specific value to “score,” that weighs our busyness conclusion accordingly.

Deven focused on the “Places” model, working through how to use the votes attributed to a certain place to calculate a score, as well as how to display these scores in a useful way. At first, the busyness score was a simple average of all votes. Eventually, this developed into a weighted average, with the algorithm valuing more recent votes proportionally more than older votes. This meant that if someone had just voted that the gym was busy, it would have more weight in the calculation than a vote for empty cast 2 hours before. We continued thinking of ways to improve this algorithm but for now it seems to be the most logical.

The next addition to the Places functionality was allowing the user to select a timespan they were interested in. At the moment we just have a few options for time spans we think would be the most relevant, although the user can manually enter any number of minutes they are interested in. Choosing the “past hour” option, for example, re-scores the current place’s busyness score using only votes within the past hour, weighting more recent votes within that period more heavily.

After working out how to deal with different time periods, we used an open source javascript graphing script to show our data on simple graph, which changed as the timespan changed. Because we did not have much experience with javascript, using this seemed like a good temporary solution, although ideally we would like to come up with our own way to display the votes. The final addition to the Places model was a color code for different scores, to create a simple and visual way to quickly see how busy different places are at a glance.

The information about different places is split between two pages, the first being a simple grid showing each place name, it’s current score (by default using votes within the past hour), and a color representing it’s score. By clicking on a certain place name, you are taken to the second type of page which shows a more detailed summary of the place, including the graph and buttons to change what timespan you want the scoring algorithm to use.