Yale Rides: Developing a Website for Travel Coordinating

Abstract:

Most of the Yale community does not have a means of transportation. Throughout most of the year, this is not an obstacle as most students rarely travel far from Yale, excluding trips to New York. The beginning and end of breaks, however, result in many students traveling to airports. During these times, traveling could be greatly improved by ridesharing. Unfortunately, there is currently no good method for finding someone to share a ride with. My project aims to solve this problem by providing a simple, easy-to-use website for users to connect with other travelers.

The website’s core functionality is centered around a calendar view where users can add their own travel plans or request to join another’s plan. When attempting to find a match, the website will send emails on the user’s behalf in order to put people in contact while still allowing privacy until a match has been made. The emails will be sent from a Gmail that I connected as an email host in Django. The website requires a Yale email to create an account, and users must be signed in to access all other areas of the site. There are also intuitive displays and buttons which enable users to create, delete, request, and cancel rides with ease.

I used Django as the basis of my web framework. As a result, I coded in Python. Because this is a website, I naturally also used HTML, JavaScript, and CSS as well. The website, at the time of writing, is hosted on Heroku at http://yalerides.herokuapp.com.

Introduction:

While everyone may not need the services provided by my project, a large number of people could benefit from its use. The problem with the alternative solutions to getting to the airport is that there are not really any good ones. For instance, consider a student attempting to fly out of Hartford’s airport. There are currently only two real options available to most students. The first option is to use a service such as CT Limo or GO airport shuttle. These services are used by many students despite regularly providing bad service at a relatively high cost for what you receive because there is no good alternative. The other suboptimal option students have is to rent a car or request an Uber. While these services allow students to avoid poor service, they are now forced to pay a higher rate.

My solution takes advantage of the fact that many students will be flying out on the same day at roughly the same time. Any student that has used a service like CT Limo will have noticed that many of the other patrons riding in the van are fellow Yalies. If the students had been able to coordinate ahead of time, they could have simply requested an Uber to take them to the airport at an agreed upon time. Sharing an Uber allows students to take of advantage of the excellent service Uber provides while cutting down on the cost to get to the airport. Sharing an Uber or
rental car with only one person is already significantly cheaper than CT Limo, and if shared with more students the savings are simply increased.

If ridesharing is such a simple solution, why don’t more students do it? This is where my solution comes in. Currently, there is no good method for finding people with similar travel plans. You can attempt to coordinate with friends, but this requires knowing all of your friends’ travel plans. Furthermore, even with all that information, the pool of potential fellow travelers is far smaller than the entire Yale community. Some try to expand this pool by posting on social media such as Facebook. While helpful, this is still far from ideal. Facebook was not designed to help people coordinate rides and unsurprisingly does poorly at this task. In my solution, the website is designed specifically to help Yalies rideshare. Consequently, this expands the pool to potentially include all members of the Yale community who would be interested in sharing a ride.

**Approach:**

The goal of providing Yalies with a convenient way to rideshare could be accomplished in several ways. The first approach I considered was having users input information about their travel plans, and then behind the scenes I would match the users together. There were many different design decisions that made me hesitant to use this approach such as what to do if there is a potential four-person match, but only two accept. Are the two that paired now out of the matchmaking system? If not, how are they handled in future matchings? What if they are paired with two new people, but only one of the original pair accepts? As people will likely to continue adding travel plans, how far in advance should the system attempt to create matches? How regularly? While I struggled with these questions, I ultimately decided against a behind the scenes matching system for different reasons.

I wanted people to have the freedom to prioritize the aspects most important to them and base travel plans around potential rides. For some, walking from Morse to Phelps Gate might be worth leaving an hour later. For others, leaving an hour earlier might be worth not hauling their luggage across campus. No matter what matching system I implemented, it would have to determine those tradeoffs when assigning matchings. Users determining their own matches ensures their preferences are accounted for as much as possible. For users to determine their matches, there would need to be a public list of available matches. This gives users the added benefit of being able to base travel plans around already created ride appointments. For example, when determining between two flights to Hartford, a user could pick the flight with an arrival time that would allow them to share a ride with the person that had already created an appointment.

To implement this approach, I decided that a web based service would be best. There is no reason that this should be a real form of software because you would want to use it immediately on any computer with internet. Also, I would never install a program for something like this. I considered creating a phone app. However, because the core features of creating and browsing rides would be much easier to do on a computer, I opted to focus on a website.

**Resources:**
As this is a class project, I wanted to learn as much as possible and use resources that I thought I would like to know in the future. Additionally, I wanted my completed project to be well constructed and actually be used by the Yale community. With these two things in mind, I determined what resources to use. I had heard good things about Django as a web framework. Django also had the added benefit of being coded with Python which I wanted to improve my skills on. For hosting, Heroku was recommended to me. Offering free hosting until a certain usage threshold, Heroku seemed like a hosting service I would likely use in the future, and thus a good candidate for this project.

**Implementation Issues:**

*Creating Appointments:* The premise of the entire project is that some users will create ride appointments, and other users will request to join that ride appointment. Thus, appointment creation is a fundamental part of site and one of the first things I worked on. Some added difficulty was that I was dealing with datetime. In order to perform operations such as sorting the appointments, displaying them in a uniform format, or display them on the calendar, they would need to be inputted in an exact format. In order help people pick a time and ensure that the input format was satisfied, I used the following widget.

*Calendar Display:* After creating appointments, I wanted a convenient and aesthetically pleasing way to display them. I found a nice calendar view. While easy to display a blank calendar, incorporating the created appointments required a good deal of work. However, it was useful because it forced me to learn more about AJAX and Django encoders.

*Tracking Objects:* This entire process was quite difficult. On my previous coursework and projects, I would have information or values and a function would simply need to be passed them. However, with the way the calendar view works, it was initially tough to have the correct interaction happen after clicking on an appointment. Later, when I was working with request
emails, I ran into a similar problem. I needed the email to have an accept and a decline link. So, I needed a way to generate unique links, but still know which link was for accept and which link was for decline. I eventually found a way to do this with Django’s signer function.

Requiring Sign In: A had created a user model that was working well. The registration and login/logout functions were working fine. However, I wanted to require someone to be logged in to see the website and follow the accept/decline links sent to the user’s email. This lead to some of the toughest to solve problems. My first thought was to block all pages using a middleware. This solution was nice because I could exclude certain urls such as creating an account from being part of the blocked pages. The problem with this implementation was that after requiring a user to login, it wouldn’t know how to redirect them to the page they were initially attempting to reach. To solve this problem, I abandoned the middleware approach and made decorators for all views that needed to be protected. The decorator would check if the person was logged in. If not, it would display the login page, but never actually change or redirect the url. After logging in, the correct page would simply load because the decorator wouldn’t be triggered.

Final Result:

Users are required to make accounts in order to access most pages of the website. It is also required because most pages display information relevant to the specific user. Here is the login page. If it is a new user or someone who forgot their password, they can follow the links below.
The calendar view allows people to see all unfilled appointments for the relevant time range. All appointments are clickable and will take you to a page with specific information and options. To reduce clutter, if an appointment is full, it will not be displayed on the calendar. Whenever the user does an action where confirmation would be nice to see such as request a ride or deleting/creating an appointment, a message like the green one below will display relevant information.

If a day was too crowded, a user could switch to the week or list view for a better view of what appointments are available.
<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Details</th>
<th>Creator Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>May 9, 2017</td>
<td>1:45pm - 3:00pm</td>
<td>To: Bradley International Airport From: AKW  # of people: 0/4</td>
<td><a href="mailto:danielyoungblut223@gmail.com">danielyoungblut223@gmail.com</a></td>
</tr>
<tr>
<td>May 11, 2017</td>
<td>10:45am - 12:45pm</td>
<td>To: Bradley International Airport From: JE  # of people: 0/3</td>
<td><a href="mailto:danielyoungblut223@gmail.com">danielyoungblut223@gmail.com</a></td>
</tr>
</tbody>
</table>

View of a specific appointment. As the creator of this appointment, I can see who is currently included, email all other participants (currently would just display message saying there is no one to email), or delete the appointment.
The same appointment as above but viewed from a different user. This user cannot see who is currently in the group or delete the appointment. They can, however, initiate a ride request by writing a message to the creator of the appointment and hitting the send button below.

The creator of the appointment would then receive an email like this. There is a link to view the appointment that the person wants to join. The email also includes the message the requester wrote and two links for the creator. After this person clicks on a link, the requester will receive an email notifying them of the decision. If the person accepts, the requester will be added to the appointment and things will be updated accordingly.
After being accepted, the user can see the current people in the appointment, email all the other people in the appointment, and has the option to delete his reservation.

The profile page lets users quickly check the appointments they are involved in, and if necessary, change their password. Because there is a link to the appointment and only your appointments are displayed, this is the most convenient way to access your appointments.

Future Work:

While this current implementation satisfies the initial goal of providing a convenient, easy-to-use way for travelers to connect and rideshare, there are several features that I want to add. For starters, a phone app would be a nice addition. It would mostly be for checking your appointments and providing a messenger service, but that would still be nice. It would also let me practice making a phone app.
Connecting to APIs would be another nice feature. For instance, a calendar API would be very useful. Whenever you create or are accepted to an appointment it could add it to your calendar. Or connecting locations to google maps would also be cool.

The most important future work though is responding to users’ feedback on this project. There are many design choices that I made that may not be popular to most users. Users may want to see the name of the appointment creator before they request. Or maybe they want people to have ratings to discourge bad behavior like being late or canceling. I made decisions based on what I as a user would want, but I am only one user so a lot of the future work will depend on what users request.

**Acknowledgements:**

I would like to thank my advisor, Holly Rushmeier, who met with me weekly providing valuable feedback and suggestions.