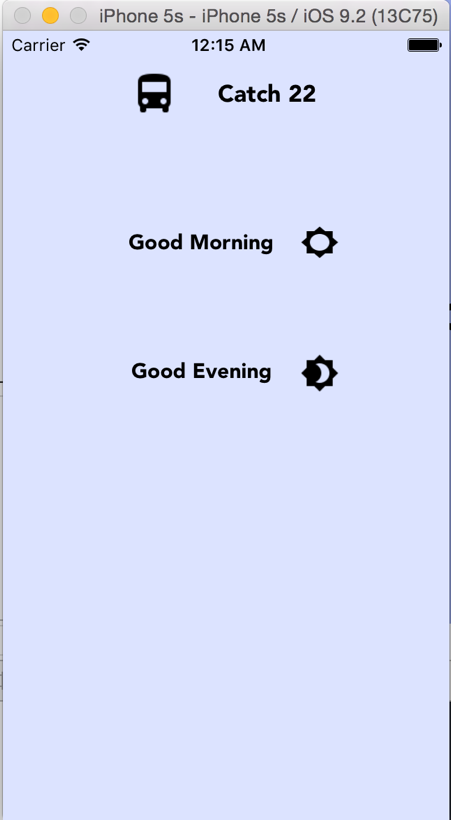
**Final Project Documentation**

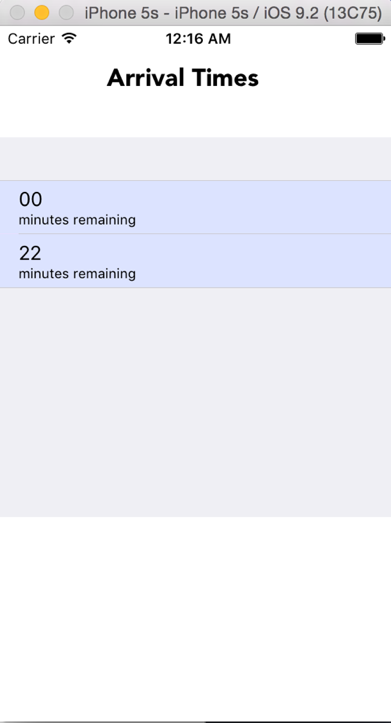
**By: Aemen Hussain**

The catch 22 app is essentially a stripped down CTA bus tracker specific to my daily needs. It allows users to get upcoming times at the touch of a button without having to scroll through bus options. Additionally, it causes users to actively think about scheduling their commutes and completing their necessary preparations by allowing them to select a time and set warnings for the bus arriving at that time. Personally, this would be helpful to me because many times when I am getting ready in the morning, I will lose track of time and end up missing my bus by a matter of minutes. I will usually open up the bus tracker when I am on my way out the door because I don’t like to deal with the redundancy of selecting the route and stop again and again. I do wish the API was able to provide arrival times further into the future so that scheduling had more flexibility but I don’t think that’s something I could have addressed on my end.

The first screen will present two options in button format for users to choose either the morning mode or the evening mode. The morning mode will select the morning stop (Bus stop #22 Clark & Wrightwood). The evening mode will return arrival times for 22 at Hubbard and Dearborn. The API request URL is selected based on which mode the user chooses. The app will then make a request to the CTA bus tracker API, get a response, parse that data into a string of arrival times and calculate the time difference between the current time and the arrival time. It will then store the differences in time into a globally accessible arrival time array.



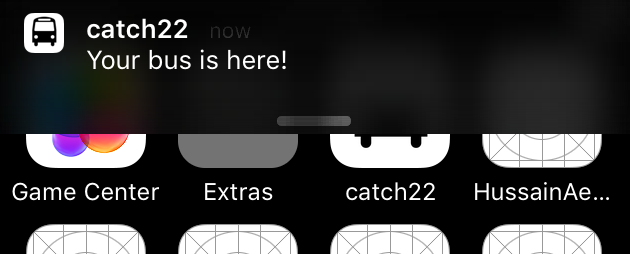
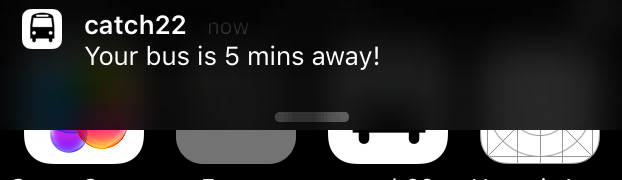
This view will now segue into the view which contains a table view of all of the times returned by the CTA API. The table view is filled out before the view finishes loading based on the values in the global array containing a string of times.



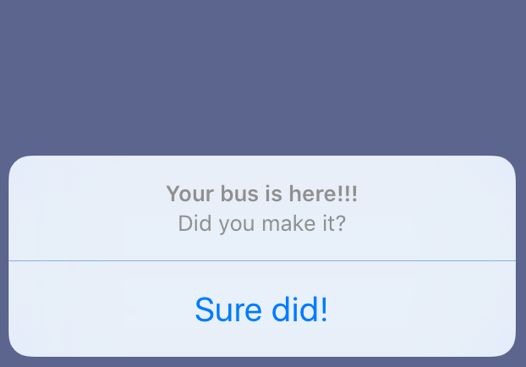
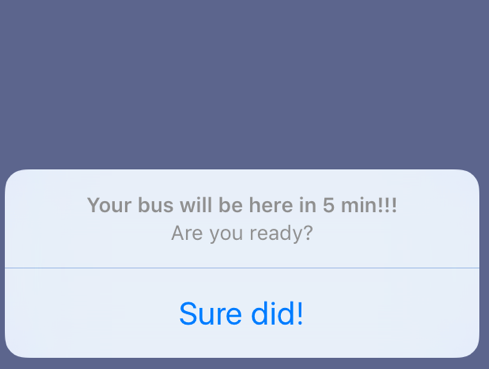
The user will be able to select the desired time displayed in the table cell and this will then segue into the following view controller which displays a countdown to the arrival time. The countdown is performed using NSTimer which keeps track of when to decrement the value displayed in the label. This value is decremented by first converting it to an integer and then displaying the integer value in the label text field cast as a string.



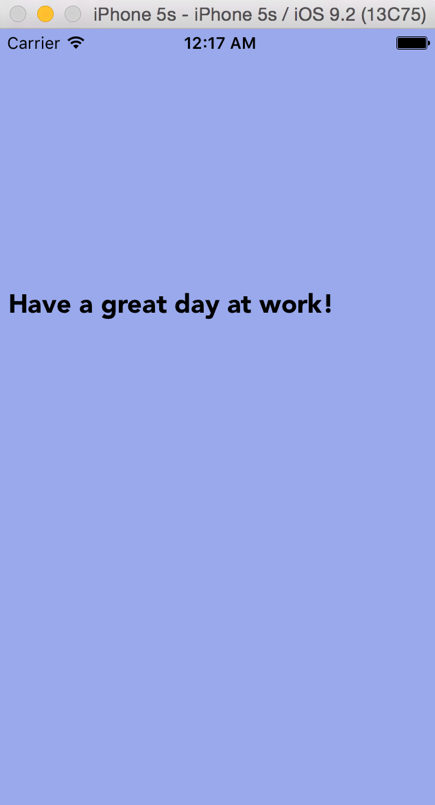
If a user chooses to leave the app, they will be presented with a push notification a) displaying a 5 minute warning and b) a notification informing the user that their bus has arrived. These notifications were set up by using UILocalNotifications. Note that the user should be prompted for permission to send notifications when they first begin using the app.



If a user keeps the app open then they will receive alert notifications for 5 min warnings or arrival time alert notifications. These are the same pop up alerts we learned to use in class.



The user has the option to acknowledge and dismiss them. The countdown screen also contains two buttons that allow the user to either reschedule by going to the home screen and re-prompting an API request by selecting their mode once again. Or users can confirm that they have caught the bus at which point they are directed to the greeting page which displays text depending on which mode the user selected (morning or evening) on the home screen.



**B. Final Project Discussion.**

Building this app was a great learning experience. The reason I chose this application concept is because it’s unlike anything I’ve done before so I knew it would be challenging for me. I’ve never built a real app before that does more than display the text “hello world” and I’ve never used an API. Additionally, I was able to familiarize myself a bit more with the storyboard builder and build off of the concepts we learned in class. Digging into the app was initially daunting but after spending several hours doing research and finding YouTube videos, articles and documentation that would help explain certain fundamental concepts, I was able to fill the gaps in my fundamental knowledge.

This app begins by using the CTA Bus tracker API. In order to use this API, I had to do some research on understanding and implementing NSURL connections. At first, I tried a far lengthier approach which took me hours to get working. When I finally got my API calls working, I was unsure of how to handle the response. I was not sure what kind of format it would be in and how I could manipulate that response. I did not know how to actually call the parsing methods on the XML response and format the response mostly because I was unsure of what type of data types I was working with. After doing some research and trying to understand NSURL data types, I found a much simpler approach that required less methods and chose to use this approach as the one you will see in my app. I used printing to test output into the console to track my progress. Additionally, finding the stop ids required some extra researching. The CTA API docs were not very detailed (at least for a beginner) so it took a lot of guess and check to figure out if I was implementing this API correctly.

Once I got the API calls working and recognized what type of data type the XML response was stored in, I began doing research on XML parsing methods. Initially I wanted to use a library I found on Github that seemed reliable, but wanted to minimize the number of external dependencies in this app and learn the code myself. Getting the XML parsing methods set up was a pretty straight forward process and did not take too much time. What was more tedious was formatting the response times so that they match the format returned by NSDate. Once I extracted the times from the response, I decided I wanted to display the time difference (in minutes) rather than displaying the actual arrival time. I figured displaying the time difference between the current time and the arrival time would also be a computation that would be necessary for the countdown view and alerts I was planning on implemented. So I found that the time returned by the API call and the time returned by NSDate had different formats. I was able to edit the strings so that the formats matched and then found a method to convert the strings into dates so that I could calculate the arrival time as minutes remaining instead of displaying the actual arrival time.

One of the biggest issues I had on this page was misunderstand how to use a segue properly. I found that I was calling the segue to the next page under the button action controller. I learned the hard way, that I should have called the segue programmatically after all of the methods in this view controllers had completed running. By calling the segue from the button action, I was actually switching view controllers before my response was received, parsed and put into an array of strings which was why my list view controller was appearing with a blank table. I even saw in the simulator that my app was already on the following view controller and the console was still logging the output from all of my functions attached to the previous view controller. When I finally implemented the segue in the correct part of the program, I was able to successfully pass the data to the list view controller.

The List View Controller also took a good deal of thought. It was different from the navigation controller we used previously in class because I did not want the entire view to be a list. I explicitly wanted a view controller with a table in it as a sub-view. I learned how to implement the table view delegate and fill the table with the desired data by the time the view completed loading. I created a table with one type of dynamic cells that called segues when touched that would pass the time displayed in that cell in the table to the countdown view. This will allow the user to interact with the table and select which time they would like to set up notifications for.

I anticipated the countdown view to be simple but it required learning more new features than I had expected. First I had to learn how to implement a timer using NSTimer along with the update method to accompany it. The count down view involved learning how UILocalNotifications work. Learning how to set up local notifications was a bit challenging but I found very helpful docs online to guide me through the process. I decided to not include sound alerts in the notifications and set up banner notifications only. I set up two notifications. One for 5 minute warnings (if the arrival time was greater than 5 minutes) and the other was a warning alerting the user that the bus has arrived to the stop. These notifications are only set up to appear if the user has the app running in the background. Additionally, the app should ask the user for permissions to send notifications before actually sending notifications. I had to alter the app delegate so that If the app is running in the foreground, the app will display alert notifications that the user can dismiss similar to the alerts we learned to create in class. I also set up two possible segues from the buttons visible on the count down view controller. One to take you to the home screen where you can resubmit an API request and schedule a new bus timer. The other option is to complete the process with the greeting view controller. One bug I found while testing my app was that if I attempted to reschedule a bus, my app would display the old request times in the table with the new times appended to the end of the table. I resolved this issue by clearing the global table time array contents after a time was selected by setting it to an empty array. This way, when a new request was made, the old values did not propagate in this round and thus the table view only contained the fresh arrival times.

One of the major limitations in this app is that I used hardcoded values for the CTA stops to simplify the API request formatting. So the stops used in this app are particular to the stops I need. In a future iteration I would like to be able give the user the ability to customize their morning and evening stops. Originally, I wanted to give implement more features to make this app more of a hub for productivity. Also, I feel that the visual design is lacking. I would like to learn how to make the app more visually appealing and give it a more elegant design. One edge case I did not have the chance to address in my code was when the API returns a response of 0 minutes – originally I wanted to change that text to display “approaching” rather than 0 minutes. I did not have time to handle this case in my code. Additionally, I wanted to clean up my code a bit by creating separate classes that handle different functions, in order to promote code re-use. Currently I am not very familiar with swift or object oriented programming so I could not implement this well in the time given. One other issue I was not able to address was how the display alerts are displayed when the app is in the foreground. If I do not dismiss the 5 min alert until after the timer has counted down to 0, then the bus has arrived alert does not seem to be prompted. I’m not sure what I could change in the logic of my code to change that.

Overall, this was an extremely fun project to work of fun. In terms of my learning style, I have found that I learn best by doing. Doing this was challenging because a lot of it involved tasks I have never done before in any of my classes thus far at DePaul so I truly enjoyed this experience. This app did not turn out as perfect as I would have hoped but I think if I had more time to learn, I would have been able to improve it significantly by testing for and addressing more edge cases.