Accomplished Functionality

The BusLinc app has 3 main pieces of functionality: Routes, Directions, and Favorites. The Routes function displays static routes and schedules. The Directions function provides the user with the means to map directions to locations provided by the user. The Favorites function lets the user bookmark locations and routes.

When the app first starts, a splash screen is shown. While this splash screen is shown, in the background the list of routes and route IDs is being pulled from the server and dynamically building the route list. A list view is displayed with Routes, Directions, and Favorites.

Clicking Routes will bring up another list view populated with all available routes. Clicking a route name brings up a Google maps view with the route, bus stops, and buses labeled. A switch button at the bottom of this view switches between the text based route information and the map view. The static schedule shows the arrival times of buses at locations. By selecting a bus stop, a little text box will appear over the bus stop with the ETA of the next bus to that location.

When Directions is selected, a view will be displayed that contains a search box at the top and a map view. In the search box, the user is able to input keywords, such as 'pizza,' 'taco,' 'burger king,' etc. The server team uses Yahoo API services, so the user is able to input any data that you would expect a search engine to handle. The top 5 results a returned and plotted on the map. The search feature also can handle addresses and this only plots one result.

The user can also hold the map for 3 seconds and a pin will drop. This allows the user to input a location anywhere. When the user selects a dropped pin or a search pin, a new view is displayed with more information about the pin. Such information includes address, phone number, distance from current location, bookmark location, and plot a route from the current location to the selected pin.

The route that is drawn from the current location and the selected pin contains the bus stop that the user should get on and off. This route that is returned from the server has a walking path to the bus stop, the bus route the user will take to the bus stop, and the walking path from the bus stop to the final location.

Selecting Favorites from the main menu will be a quick way to reselect favorite routes. A list will be displayed of bookmarked routes and destinations. On the route view, a bookmark button is on the toolbar, allowing for the user to easily bookmark routes. For destinations, there is a bookmark button on the view that contains other information about the selected destination. The user can also delete favorites by selecting the 'edit' button at the top of the view.

Development Practices

Our group divided up tasks, such that Leonard focused on testing, Michael worked on implementing new server functionality, and Andy and Luke worked on memory management and other functionality. We do not have any unique release process other than if it compiles and isn't too buggy, commit it.

A piece of functionality that our group has is A-to-B directions. The user would select a pin on the map and the app would draw the route from the current location to the selected destination pin. We have the main controller for the Directions feature called *DirectionViewController*. When the pin is selected, a callout appears over the annotation which

allows the user to click this callout and see more information about this pin. Once the pin is selected, the *SearchDetailsController* is pushed onto the navigation stack. The new view just has some labels and buttons that let the user see the address, phone number, and distance of the pin. One of the buttons says 'Directions to Pin.' A big obstacle is how to figure out how this button communicates back to the previous view with a response.

It is easy to call other methods and return data, but how do you return data from a soon-to-be dealloced class? I created variables in the *DirectionViewController* class and would try to link them somehow to the *SearchDetailsController* view, but that was impossible because the referenced value would be dealloced before it returned properly. After much searching, it seems that what was needed is to create the *SearchDetailsController* a delegate that the previous class calls.

DirectionViewController creates an instance of the delegate class, *SearchViewController*, and we now have access to the variables and methods of the class. So, when the user hits the 'Directions to Pin' button, we return a success boolean back to the parent class and pop the view from the navigation stack. Once this boolean is returned, we start the parser to get the directions from the server using the coordinates of the current location and the destination pin.

Application Quality

Testing

The initial idea behind testing was using a script to run some test cases automatically but because the scope of the project involves real-time data, it is impossible to generate an exhaustive set of test cases. With that, it is difficult to determine what data output is correct in the test oracle in terms of code. The team decided to use a more less time consuming form of testing, which is manual testing at the field testing level.

The application was also tested by parts as it was developed, mainly using the function NSLog which is the Objective-C's equivalent of C's printf function. This allowed us to monitor the data that was both sent and received by the application during run-time. Since each code section is separate module and will be able to function on it's own, this development technique was viable because integration of the components was not an issue. Additionally, the team used source control so changes are recorded and code can be rolled back in order to see changes that introduced errors.

The Apple development environment, Xcode, has several built-in testing tools to help catch errors, such as memory leaks. These tools were used throughout the entire development cycle to aid in reducing memory leaks and resulting in a more efficient application during run time.

The application features two primary functions, known as Routes and Directions. Routes uses essentially the same data as the web application currently run by Star Tran, therefore the most basic testing method was to ensure that our application displayed data that was the same as the one on the web application. This is to determine if the map drawing scheme is correctly written on the application. As for the arrival times at each bus stop, it was difficult to determine its accuracy as the estimated time of arrival at each bus stop are data received from the server, the times are not too far off from the actual arrival time in reality, with only about one to five

minutes plus the ETA shown on the application. It was also manually tested by having a team member standing at a bus stop to see if the bus arrives as shown on the application.

As for directions, this involved a more complex testing scheme that is divided into a few stages. Firstly we needed to determine if the query results returned by the server was accurate. The most frequent query we tested upon was “Burger King”. In the early stages the results returned by the server were not all Burger Kings, and that was the first testing phase of Directions. Then we moved on to checking the XML data returned by the server when we made a query asking for directions to a particular destination and the instructions of where the user needs to walk to and which bus to take was displayed on the map in the application. However, due to the inaccurate data received from the server it was not possible to manually field test it, as the directions returned required us to walk through buildings a times as well as getting on and off the bus at the same station. The testing for this feature of the application involves working closely with the server team to ensure both our data outputs are correct.

Usability

The application has a simple user interface, with the main menu only having three options known as Routes, Directions and Favorites. The simplicity of the design was aimed at having a lower learning curve for new users to use this application as the menu options are quite self-explanatory. Routes features the same functions as the one of the web-application provided by Star Tran. All the user needs to do is to selective a route from a list and the map of the route and the current bus locations will be displayed.

As for Directions, the usage is simple and straightforward. The user will only need to input a search query such as an address or a name of a location, and the application will throw down some pins on the map close to the user’s current location. The user may then select one of the pins and has the option to call the phone number at the destination or request for directions to it. Instead of using the search query, the user may also hold down on a point on the map, and the application will drop a pin on that location.The user may then also request for directions to that location.

Favorites allows the user to store a particular route and a previously requested direction instructions. It is essentially a feature designed for frequent users experienced in using this application.

Accessibility

The application depends heavily on the server end as well as Wi-Fi or cellular data for the IPhone. The server response time will determine part of the processing time during the application’s rune time. The data accuracy displayed will be based on what is returned from the server. The Routes feature should not be greatly affected by this, as the application will dynamically retrieve the route IDs and build the routes as the application starts up. The routes should not change that much but even if Star Tran decides to change the routes, the application should still display the maps correctly if the XML data received from the server is in the same format as before.

The Directions feature will depend heavily on DAPHNE. The queries and instructions for directions will depend on what the server returns. As of this time the Directions feature is not fully operational but as soon as the server improves the data the application should be able to

display it correctly. Further work needs to be done on this to improve the quality of the output as well.

Risk Mitigation

App Store Acceptance

Description:
The team needs to adhere to the iOS Human Interface Guidelines to ensure that the application gets accepted to the App Store. This is key so that the app can be distributed to the public.
Contingency Plan:
Reviewing the iOS Human Interface Guidelines to be sure that the app meets all the criteria is key. This includes “Human Interface Principles”, “iOS UI Element Usage Guidelines”, “iOS Technology Usage Guidelines”, and “Custom Icon and Image Creation Guidelines”. Again, consulting Ian Cottingham before App Store submission will aid in these processes.

Performance

Description:
Performance risks include load time, responsiveness, and memory leaks. The team can see problems in the future regarding load times with poor 3G or Wifi coverage and poor server upload speed even with a good connection.
Contingency Plan:
Inefficiencies in the code for drawing routes might also need to be improved to ensure the app runs smoothly. Fixing memory leaks early instead of leaving to the end of the semester will make things less of a problem. We will work closely with the server team to see if we can make any optimization improvements in the data sent over the network, or help identify any hardware/infrastructure bottlenecks.

Sever Downtime - Biggest Risk

Description:
If the server goes down or is not reachable from the iPhone or simulator then our app cannot work and development can’t continue until things are back online. The server team also depends on Yahoo API services, which can also go down.
Contingency Plan:
The team will ask the server team to let everyone know when there will be downtime for changes or upgrades to the server. The team can also trap static data as local files to simulate the XML over the network so we can continue development even if the servers are down or offline.

Conclusion

This project was successful because of teamwork and motivation to make a better product than the other phone teams. Since we also had other members focusing on various parts of the project, more was accomplished. There was always someone working on testing, functionality, deploying on the iPad, etc.

We learned a lot about XCode, Objective-C, memory management, how to build usable interfaces, the iPhone, and how to work as a team. This project was very interesting because mobile devices are a unique platform due to the size, usability, and capabilities. Because of this we had to change our mindset on how to approach functionality and usability.

This solution can be improved by using iAds in innovative ways, such as mobile coupons. Also, dynamic alerts can be used to give users a prompt if they are going to miss their bus.