TAMUber Student Interface Final Report

Summary

Our project implements a flow of pages and handles the actions needed for an uber-like reservation system of the self-driving golf carts on campus. Most of the points in the program where carts would actually be communicated with remain stubbed out because our code relies on ROS messages that the carts have not been updated to handle. Due to this, our project serves as a foundation for future work on TAMUber where the carts are updated to be able to communicate with our application.

The customer's primary need was to have an extensible basis on which to build the fully fledged TAMUber service. This need has been met by ensuring only simple changes are required once the necessary changes on the cart's side are made available to convert the project into a fully working uber system. The required changes on the cart have been clearly described in the user guide to make this transition easy for the customer.

User Stories

1. Be able to call a TAMUber to a location near me
    - This feature was given two points because it's important as a foundation for the application
    - Feature has been implemented, when real carts are active it will send the gps location of the pickup to direct the cart there
    - Initially we expected to use route names to specify the location, but the customer informed us that the carts will know how to get to a location given just the gps coordinates, and know if it's on their route or not
2. Be able to select a route on the map
    - This feature was given two points because while it is important for the system to run fully, it's not a very large feature
    - Feature is implemented to allow users to select their route easily and see it on the map before scheduling to make sure it's the right one
    - This feature initially expected the user to be able to click on the route shown in the map widget, but due to the number of routes and the limited area they would overlap too much for this to be useful so a dropdown below the widget is used which displays the selected route on the map widget
3. Be able to specify desired seats and handicapped access
    - This feature was given one point because it adds convenience and increases efficiency of TAMUber by allowing multiple people per trip
    - Feature is implemented but requires carts to provide seating limit and specify handicap capability

    - Story was added after a follow up visit with the customer where the need for this functionality was expressed
4. Show me the ETA of the cart to the pickup location
    - This feature was given two points as it's a good convenience feature to let people plan their time as they need to when using the service because they can know how long they'll be waiting
    - Feature is implemented but relies on an having an accurate average speed of the carts to compute time from current waypoint to final waypoint
    - Initially was going to use the walking estimate from an approximate google maps route, but this was scrapped since we have the actual waypoints available and need not approximate that aspect.
5. Show me the routes available from all carts
    - This feature was given three points as it's critical to allow the system to run fully
    - Has been implemented
    - Initially involved getting the routes stored on the server, but based on how the carts are set up it was altered to collect routes from the carts using ROS and then display them to the user
6. Be able to cancel my trip prior to pickup if needed
    - This feature was given one point as it's more of an additional feature, not required for the base functionality of the system
    - This feature was not implemented
    - Difficult to implement when the carts can't be modified to respond to cancellation messages soon enough for this class

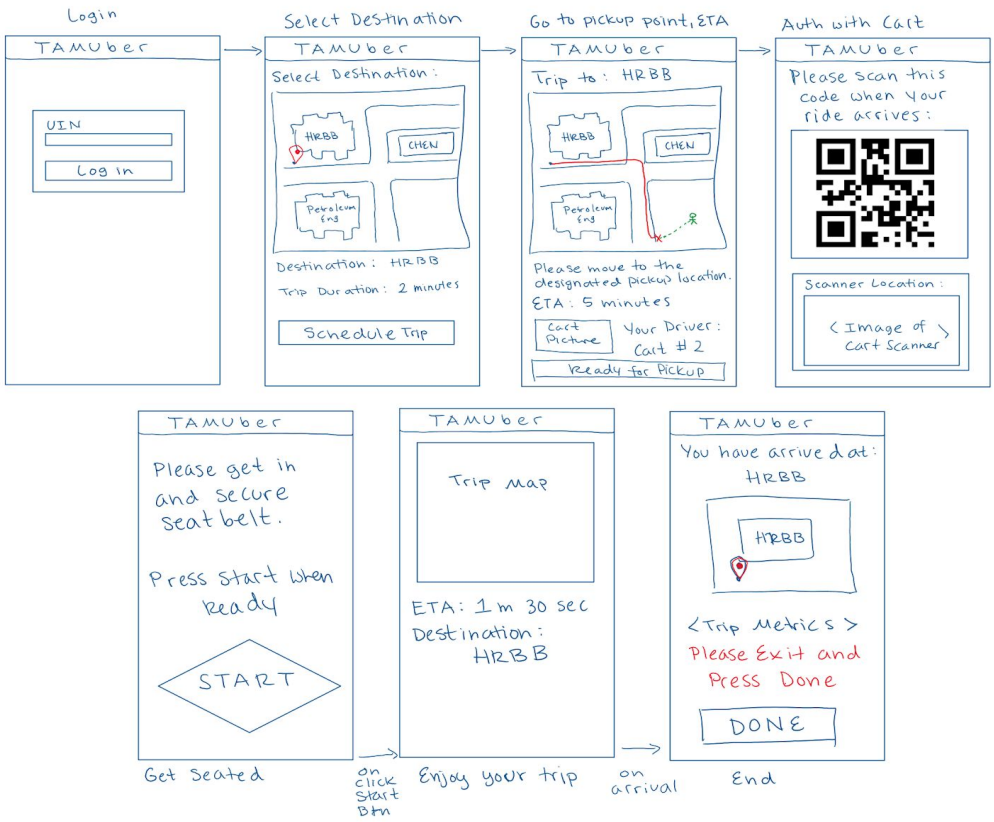
Roles

Scrum Master: Jordan Taylor
Product Owner: Caleb Stephens
Roles did not change throughout the project, however the duties of the scrum master extended to managing all pull requests for the git repository.

Agile Iterations

Iteration 0

In this iteration the team met with Srikanth Saripalli to discuss his vision for the site. The GitHub repository and Pivotal Tracker were set up and an initial set of user stories and a low-fi UI mockup (shown below) were created. It was established that since this site will primarily be used on mobile devices, the UI should be kept simple. The final set of user stories can be seen in the User Stories section of this report.



Iteration 1

In this iteration the team began doing research on how to interact with the cart via ROS nodes and how to use the Google Maps API. By the end of the iteration, the site was able to display a route on a map from a static set of GPS coordinates. The architecture of the code was discussed and an initial plan was made for what models, view and controllers were needed. After meeting with the customer, it was decided that the authentication provided by the QR code page in the initial low-fi UI mock-up could be satisfied by a button the user presses when they are ready to start their ride, so the QR code screen was eliminated from the mock-up.

Iteration 2

In this iteration the team was able to display the pickup point of the user's selected route on a map. In addition to this, the user could view directions from their current location to the pickup point. The routes were made to dynamically display from the route model rather than from a static array of GPS coordinates. It was decided that roslibjs would be used to interact with the cart and the team was able to use this to get data from a ROS node.

Iteration 3

In this iteration the method to select and view a route was changed from a drop-down menu to a table. Rather than seeing every route on the map at once, the user could click ‘view’ on a route in the table and it would appear on the map, highlight the route being viewed in the table and allow the ‘start trip’ button to appear. The team was also able to get a marker on a map to follow the location of the cart published by a ROS node, and progress was made on how to calculate the estimated time until the cart’s arrival at a destination.

Iteration 4

In this iteration the team learned that routes would be stored in text files on the carts and different carts may not be able to go on the same routes, so the role of the cart route model changed. Instead of keeping a store of routes that all carts could run, the routes that a cart can go on were read from the carts and sent to the backend, where they were filtered based on cart availability and user requirements, and then they were sent to the view to be displayed on the ‘select a route’ page. Once the user selected a route, the route information was loaded into the route model and added to the user’s trip. In addition to this, logic for marking the cart in use was added, including a timer that unmarks carts if they are left alone for more than five minutes. The estimated time to arrival was added to the views and by the customer’s request, an extra screen in between the splash page and the select a route page was created to allow users to specify how many seats they need and whether they need handicap access.

Customer Meetings

Meeting 1 (Feb 19, 4:00 pm):

The client described the project to us and laid out the basic requirements of our software. He demonstrated the autonomous carts and provided some basic ROSBridge tutorials for us to work with, knowing we had minimal prior experience with this platform. We recorded the requirements and formed some basic user stories from the information the client provided.

Meeting 2 (March 2, 4:00 pm):

We showed the client our implementation of the first and second user stories, and asked for feedback early so as to make sure we were meeting his expectations. At this point, we still had no ROSBridge code added to our application. The client provided contact information for two graduate students who are working on the autonomous carts who would give us the vital information needed to communicate with the cart from our app.

Meeting 2.5 (March 5, 2:00 pm):

We met with the graduate students working on the cart. They gave us the IP address of the cart, as well as the types of data the cart would be publishing. They also showed us which functions we would need to use to communicate with the cart and how we should use them. They told us that we should contact them if we needed the cart to be turned on and publishing data.

Meeting 3 (March 30, 4:30 pm):

We gave the client an update on our progress by giving him a run-through of our whole application up to that point (which consisted of user stories 1,2,5). He was satisfied with what we had thus far, and reiterated that the carts would be selecting routes from a predetermined list, and that each cart should have a different set of routes.

Meeting 3.5 (April 4, 1:00 pm):

We met the graduate students working on the cart to sort out connectivity issues with ROSBridge. The problem was that the port we were connecting to had not been forwarded. They also explained to us that we would be requesting services from the cart that have not been implemented yet. To work around this, they suggested that we would create a dummy service call with all of the parameters we would need, and they would accomodate us later.

Meeting 4 (April 20, 4:30 pm):

We met with the client and showed him our (almost) completed app. We were missing options for handicapped students, as well as an option to select the number of seats. However he was satisfied with the app as a whole. I emailed him after we had implemented the missing features and he was happy with our finished product. We could not meet for this demo as he was on travel. He asked us to create a how-to document for setting up a server with our application on it, and to add new carts to the list.

BDD/TDD Process

Where it made sense, we used Test Driven Development (TDD) and wrote tests before code. This included Cucumber tests for integration testing. Cucumber tests were mostly written for testing page navigation. A lot of the code relied on javascript and communications with carts (which wasn't complete cart-side), and we struggled to create automated testing for this as it required a different framework and a lot of mocking up of calls to the carts, both with ROS bridge code and rails methods. It ended up faster for us to just manually check that the google maps worked since it rarely changed. The cucumber testing worked for a while but for parts it requires that a cart has arrived at its destination before the interface was presented to the user to proceed to the next page, so it was difficult to figure out how to stub that part of

the framework out as it queried carts over javascript. Additionally most of the links were static and so only had to be tested once.
We used Behavior Driven Development (BDD) in this project in the form of user stories. We tracked these on pivotal, which allowed us to distribute each story and track what features are in progress, completed, or not started. Of the two, BDD and TDD, this was definitely the most useful for us. By breaking up every feature into fine grained user stories for what the user wants to be able to experience, we were able to plan out who would do what features and in which order. Each feature ideally had an easy test to determine whether or not it was completed and working. This alleviated much of the testing work since we only added features once old ones were shown to be working as intended.

Configuration Management

We used git for version management, hosting on github. Each person cloned the repository to a local repository, either on their own machine or on a designated subfolder on cloud9. The master branch on github is intended to always be the current working version of the code, and is what is pushed to Heroku for the live site. We utilized feature branches, so when someone wanted to work on a feature they would branch from master and name the branch based on their initials and which feature the branch is for. Changes would be pushed to that branch often, and when the feature is considered complete and merge conflicts are resolved, they would use Github to create a pull request to merge that branch into the master branch. Someone else would review the request and if it is accepted, it is merged into master. Github also was integrated with Semaphore, which provided an indication of whether or not each commit passed the automated tests.

Heroku

Some, though not all, of the deployments to Heroku were challenging. The team used SQLite3 for the development database, which was easy to work with. For the production database, however, PostgreSQL was used, which had a learning curve. The Heroku environment was just slightly different enough from the development environment to cause many small problems that were difficult to debug, but eventually a solution was found for each problem.

Implementation Environment

This project was based on collaboration, often simultaneous. To make it easy for remote simultaneous collaboration, we chose Cloud9 hosted on AWS. This allowed us to remotely edit the same files simultaneously, like google docs. This also allowed us to test our app from different computers instead of allowing people to access our local machines. Our environment consisted of tools described below, and ruby on rails and its dependencies. We also configured postgres on our environment because heroku doesn’t allows sqlite in production. We used git for version control and coding conflict resolution. Heroku Command Line Interface allowed us to easily push our release code to heroku where everybody can easily access the app.

Tools

roslibjs - A library for the ROS bridge which allows our application to communicate with the carts from the client pages using javascript. This is how data is requested from the carts and published to the carts. This is the bridge and so it does not require the full ROS install that the carts have, as this wouldn't be possible on heroku.

google maps - API for the map widget used on many pages of the application, this tool allowed us to easily display and add information to the map screens for the user, such as the route currently chosen on the scheduling screen.

semaphore - This tool would run our cucumber tests for us to ensure a pull request won't break any important features. It ended up not being very useful because we would have to run the program to test things not covered and tests would be run as a matter of course in testing this way before merging a pull request.

Final Interview

The client was not available for an interview as he was on travel. However, he answered a few follow-up questions via email. Below are the questions and their corresponding answers word-for-word:

1. If you could change anything about this app, what would it be?
2. What are your primary concerns in rolling the application into production?
3. Does the styling look good enough?
4. Is it "user friendly"?
5. Does it meet all of the design requirements

--It looks much better on my phone than on the iPad
-- Primary concerns are how robust would the app be. Is there a way to delete the trip. If all the carts are full, the app should say something like all the carts are full.
-- Styling is great
--I think so. I was not able to see the end trip button and currently it is not working so can't say more
--Yes

Links

Github: https://github.com/mhtee/tamuber-students
Demo: https://youtu.be/QwBh3MSPNCk