Let’s Carpool! Iteration 1

AJ Cihla, Aime Ngongang, Peter Lee, Stephanie Ku, Alex Wong, Sam Vu

Main Difficulties:

We had some substantial issues with testing and then cleaning our separate databases. Utilizing the Django automated test database ended up being extremely helpful and we didn’t have to keep clearing databases between each of our individual configurations. Corruption of dependencies and standardizing our environments cost a great deal of time but hopefully that chapter of the development process can be mitigated in the future through better means of communication (using Pivotal religiously) and general development strategies (ex. Committing often, pull often, using other git functionalities appropriately, don’t push if you aren’t sure that code works, etc.)

Proper utilization of Google Maps API was definitely coupled with a steep, yet surmountable, learning curve that took a few hours to understand. Luckily, online documentation was plentiful and very helpful for most cases. We did run into the problem of accidentally starting to use syntax and configuration details correlated with an earlier version of Google Maps. Merging android layouts, each of which required its own build configuration, was also a major issue but sitting down together and finding a hybridized solution made it all possible.

Missing Features:

The abilities to filter routes by locality or other heuristics, showing ride details, and accepting ride requests are features that we did not develop. The UI is also lacking in many respects but I think that the majority of our focus was being focused on functionality of our key features and building up a solid base of code from which to work. UI can always be changed, although it may be time-inefficient to deviate from the original design plan and have to spend time in the future modifying and refactoring our existing code.

Testing:

Backend: unit tests and functional tests are being used for 3 major server-side requests coming from our front-end application (adding route, searching for routes, and requesting a ride for a certain route).

Frontend: JUnit tests were used for functionality (adding a route, viewing map, and requesting available routes). The UI was tested using its own separate JUnit tests to make sure that interface elements were correctly sending signals for action elsewhere in the application.

Testing did not deviate from the plan although our methods did. I think this is a definite selling point of test-driven development because although our procedure means may change, the result will generally remain the same. Currently, our backend and frontend are at different levels (backend has some functionalities not utilized by frontend) but besides that we are testing all of our methods. The interface between front-end and back-end is tested for general communication but specifics of the interaction are not being tested currently.

Git: