Winter 2016 Programming Contest

Team: The_Flatlanders

School: Walt Whitman High School

Students: Joshua Engels, Rory Nevins, Joey Wood, Ari Mindell

Code Related Aspects

Correctness/Completeness

- Does it work and how well?
- Have other potential uses for the code been considered?
- Does it solve the problem for varied inputs?

Your solution works very well. We did not experience any crashes or issues with your software while it was running. The output for each package (details) seems to be very complete and correct. Your solution seems to implement all of the requirements we put forth for our package tracking interface. One thing I was searching for in admin mode was the ability to see a list of all packages that are out for delivery (I think the mechanism you have in place is to essentially look at the map, and you'll see a marker for each package that you need to individually click on). This was not a requirement, but it was a common feature in other complete solutions and seemed to make admin mode more usable.

When we launched the application interface, we were ultimately opening the site from a file, rather than an address. Was this intentional, or were we opening the site incorrectly? The difference here is that the address in the url bar for the site was file:// instead of http://. Ultimately, we were not able to test multiple users hitting the site at the same time from different machines on the network, because the site was not actually deployed. All interaction with your application was done using the machine that it was installed on.

Our initial use of the application left us scratching our head because we used the UI to create our user, and did not see a way to create an admin user. The more we thought about it, this must have been a security concern on your part, because you would be giving any user that uses the interface the ability to mark their account as admin. The inclusion of the Groovy script on the server that had the ability to create both types of users certainly answers the mail to allow us to log in as either admin or end user. This sort of consideration speaks to the level of completeness that your team achieved.

Your team should be very proud of yourselves for implementing an excellent solution to the given problem. Bravo.

Modularity/Usability

- How easily can it be integrated into the code base?
- Could it be integrated into a tool chain?
- Are there logical separations between components?

One example of excellent modularity is the user account code that could be triggered as a standalone groovy script, or could be triggered through the application to create users. You guys did a good job keeping the user admin code modular enough that it could be used again in a completely different application.

One thing to consider would be moving the code that computes distance and midpoint out of the Coordinate class and into a common utilities class. Imagine that you are a company like Uber and you are growing rapidly. A whole bunch of features are being added that will also need to calculate distance. If you keep your computations locked away in individual classes, you will discourage other people from using the same algorithms/code. If you had a static utilities class to do all math or all geo computations (e.g. GeoUtilities.groovy), you would encourage all developers that are adding additional features to use a single, common solution (instead of 10 different distance algorithms, all calculating differently).

Your code is very thin. In that respect, it is very easy to work with in terms of integrating into an existing code base. Nice job avoiding code bloat.

Maintainability/Documentation

- Were the comments sufficient and variables appropriately named?
- Could the code be understood/fixed/extended easily?

It is always appreciated when people add javadoc style comments to explain the expectations for methods. You guys did this for a lot of methods but not all of them. Since so many of them were done, it might have been cool to generate actual javadoc documentation or the Groovy equivalent (groovdoc?).

There were a handful of magic numbers found in your code (e.g. 3600 – could have been saved as a static final variable called SECONDS_PER_HOUR). Otherwise, you guys did a great job self-documenting your code with descriptive method and variable names. All of the calculation logic (midpoint, getDistance) was easy to follow and cleanly coded. The actual code blocks contained minimal comments, but the code was clean and concise enough to probably not need much at all.

I certainly felt that I could take your project and extend it (add new functionality, or update existing functionality, or borrow aspects for other projects). Code is easy to understand.

Performance/Scalability

- Does the applications performance meet explicit/implicit needs?
- Do the algorithms scale appropriately in time/memory?

From a scalability perspective, I thought it was interesting that you modeled a coordinate point as a class because each package potentially has a tremendous amount of coordinate points, depending on how long it is out for delivery. I recognize that you used the coordinate point class as a data container to hold the coordinates, as well as a location for the logic associated with computing distance and midpoint, but that could be a very large number of objects going onto the heap and you could potentially blow out your memory just from holding onto coordinate data (since each package holds an array list of all of the past points). It might make more sense to push everything out to a database so that you don't blow out memory as you scale up.

No problem was observed with the performance of your application as we sent a handful of packages through. Everything within the application remains responsive.

User Experience Aspects

Delivery

- How well were the submitted components labeled?
- Were the documentation/source/executables logically separated?

Your team organized your delivery well in terms of logically separating components. All of the web viewable code was in the htdocs directory, all of the package sending code was in Event Simulator and all of the Groovy code for your solution was in the src directory. Thanks for putting the documentation at the top level instead of burying it deep in the delivery.

That was thoughtful to include the GRE so that the user did not need to download Groovy in order to send packages (using your .bat files). It was interesting that the GRE was not pulled up to a higher level in the delivery and used to run the Groovy code in src. I suppose that this was influenced by the fact that you wanted the user to launch the application through Eclipse (and to have installed Groovy support for Eclipse), but there was still a disconnect between the delivered GRE and the fact that the user had to go install Groovy through Eclipse.

Documentation

- Was the supplied documentation easy to understand?
- Was the acceptable input format well defined?
- Was the tools output well explained?

Your team provided good documentation in the README.md file, separated into Product Documentation, Software Documentation, and Software Requirements. A text based document is

always a good choice because it doesn't assume that the user has Microsoft Office or Adobe Reader installed. Thank you for including Eclipse plug-in installation paths in your documentation. We definitely had to lean on the documentation to figure out how to get the software running as well as to figure out how to make accounts.

Build Environment

- Was it easy to build and execute the application?
- Were all details of the build explained well?

The details of how to build/execute were well explained in your documentation. Build Environment was one of the areas that turned out to be a huge differentiator between all of the projects. Ultimately, the ideal solution here would be some sort of installer, or set up script that takes care of everything, or prepackaged executable containing all dependencies. All of those minimize the amount of work that a user has to do in order to get the application up and running. We did not have any problem getting your project set up, but it required significantly more effort than some of the other projects.

User Interface

- Was the interface easy to interact with?
- Did the solution execute easily?
- Was all presented information understandable?

The user interface for your application certainly contained everything that we were looking for, particularly the integration with a mapping api to display location of packages. You guys had some features that were really nice, such as tracing the route on the maps and custom notes for packages.

One thing that was a tiny bit clunky was the Refresh Map button. It seemed like that always reset the zoom to the world level. We noticed that when we were zoomed in on a package, the package did not 'move'. To see the movement of a package, it seemed that you had to refresh the map, then zoom back into the package.

Another aspect that was slightly clunky, specifically in Admin mode, was a lack of a list of all of the packages. It seems like the only way to access a package is by clicking on it on the map, instead of seeing a list of all packages that are out for delivery. It also seems like the only way to know how many packages are out for delivery is to count the number of markers on the map.

The details that you provided for each package were excellent. No question there.

Requested Feedback

- **We used cookies in our final solution to keep track of the currently logged in user. We knew all along that this was a particularly fragile method of storing secure data, and we did our best to optimize it by storing only the username and not the password, but we understand that regardless, a user with cookies disabled would not be able to use our solution. We ended up running out of time to resolve this issue, and instead put a message at the bottom of the login screen explaining that cookies needed to be enabled. Is this solution acceptable? if not, what would have been a better way to implement it?**

It is acceptable inasmuch as it is a rapidly developed solution that works and it demonstrates the concept. Cookies would not be my preferred mechanism for a publically deployed solution on the internet, because of security concerns (cookies can be manipulated on the client side). Most back ends use sessions to keep track of the unique users interacting with the site.

- **Another bug that remained in our code after submission was the interaction with chrome web browser. We were unable to find a way to force chrome to allow our static HTML to send post requests to the servlet. Is there a way to do this without hosting everything on the actual internet?**

The problem that we were seeing on our end, when using a Chrome browser, was a 'null' value for the Origin header (just to make sure we are on the same page here). The nature of the problem seems to be coming from the fact that we were launching the website from a file (file:// was the address) instead of from a url (http://). The problem can be solved by running a web server locally, and placing the files in the web server's root directory. That way, you can use a url to access the files, such as http://localhost/mainPage.html.

- We realized upon reading your email that our installation process involved installing eclipse to be able to run the server. Should we have compiled the server and simply had an executable file for the idt user to run?

The final submissions that we saw for this contest ranged from development environments to full blow installation scripts. We did get your software up and running, but we were using a Windows virtual machine that did not already have eclipse (or any development environment) already installed, so it took a bit of work. I think it is awesome that you did deliver a project that we could import, look at the code, and compile so easily. An additional piece to your delivery that would have been very convenient for IDT would have been an executable version of your solution.