Instant Package Tracking & Reporting System (IPTRS)

Requirement Traceability Matrix

Team Game of Life

Oakton High School

February 12, 2016

Software Requirements

1. The solution shall handle multiple simultaneous GPS tracked packages sending updates.

Test result: We tested this requirement by calling logger method in the source code. When running the application, we also ran groovy simulator with multiple *.GPX files, we see logs for multiple packages are generated. Each package has a unique UUID.

2. The solution shall be easily accessible from a Windows 7 computer.

Test result: we tested this requirement by running the application from a Windows 7 machine. The application ran smoothly. We also tested the application from a Windows 8 machine. It ran well too.

3. The solution shall support an admin mode that shows all package location updates on a map.

Test result: We tested this requirement by running groovy simulator with options *`-n -c 30 -m`*. This ensures all gps routes are converted to real time and compressed within 30 minutes. Our application shows entire delivery path on the map. We also ran multiple *.GPX files from simulator, and the admin view in our application shows all packages corresponding to the GPX files.

4. The solution shall support a user mode that shows a subset of package location updates on a map.

Test result: We tested this requirement by entering selective UUIDs after starting all *.GPX files within groovy simulator. The user view only returned the packages that are associated with those UUIDs.

5. The solution shall accept a list of UUIDs in user mode to control the subset of package location updates displayed on the map.

Test result: We tested this requirement by entering selective UUIDs after starting all *.GPX files within groovy simulator. The user view only returned the packages that are associated with those UUIDs.

6. The solution shall accept name, destination, and GPS unit UUID information as HTTP query parameters on a HTTP GET of the URL path "/tracknewpackage". An example follows: GET http://127.0.0.1:8080/tracknewpackage?name=Some+Name+Here&destinationLat=42.4877185&destinationLon=-71.8249125&uuid=b0f9bb21-160f-4089-ad1c-56ae8b2d5c93

Test result: We used SoapUI to test this requirement.

7. The solution shall respond with a JSON encoded body which includes the registered uuid on an HTTP GET of the URL path "/tracknewpackage". An example follows: GET Response Body: { "ackUUID":"[b0f9bb21-160f-4089-ad1c-56ae8b2d5c93]" }

Test result: We used SoapUI to test this requirement. JSON code showed in correct format.

8. The solution shall accept a JSON encoded body which includes location, elevation, and time on a HTTP POST to the URL path "/packagetrackupdate/". An example follows: POST http://127.0.0.1:8080/packagetrackupdate/b0f9bb21-160f-4089-ad1c-56ae8b2d5c93 POST Body: {"lat":"42.4879714","lon":"-71.8250924","ele":"195.9","time":"2015-12-08T08:42:33.188-05:00"}

Test result: We used SoapUI to test this requirement. JSON code showed in correct format.

9. The solution shall accept a JSON encoded body which includes a delivered flag on a HTTP POST to the URL path "/packagetrackupdate/". An example follows: POST http://127.0.0.1:8080/packagetrackupdate/b0f9bb21-160f-4089-ad1c-56ae8b2d5c93 POST Body: {"delivered":"true"}

Test result: We used SoapUI to test this requirement. JSON code showed in correct format.

10. The solution shall calculate and display distance to destination.

Test result: We tested this requirement by calling logger method in the source code. We can see calculations in logs. We also tested this requirement by using UI.

11. The solution shall calculate and display estimated arrival time.

Test result: We tested this requirement by calling logger method in the source code. We can see calculations in logs. We also tested this requirement by using UI.