Winter 2016 Contest Software Requirements

Software Requirements – Our solution

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

Our solution used Node.js to create a RESTFul web service that stored package data from simultaneous updates

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

Our solution was to serve an html/css/javascript page at 127.0.0.1:8080 so that the computer hosting it could access the solution, but when actually rolled out could be accessed online by any computer.

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

There is a radio button in the bottom right hand corner that toggles admin mode.

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

There is a radio button in the bottom right that toggles user mode.

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

When user mode is toggled an interface appears that allows users to enter in UUID's of packages that they wish to track.

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

The node side of the solution handles this get request using the express library for node and updates an object to keep track of packages.

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]" }

The node side of the solution handles this request using the express library, acknowledging the UUID correctly if valid and returning {"ackUUID”:”false" } if the uuid is not valid.

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"}

The node side of the solution handles this request using the express library, accepting the package update and updating the object representation of the package that is being updated.

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"}

The node side of the solution handles this request using the express library, accepting the body and updating a package to the delivered state.

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

The solution uses Google’s geometry functions to calculate distance between package location and destination in Kilometers.

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

The solution uses Google’s distance matrix framework to calculate driving time to destination unless it is unable to make this calculation, and then a formula based on average speed of a package is used to give a rough estimate of time of arrival.