Uber Freight Tracker

Team:
Daniel Fialkov
Jason Wakumoto
Kevin O'Linn
Derek Wang

Client: Ricky Walker

6/8/2020

Introduction:

Uber Freight is an Uber subsidiary that connects freight drivers with enterprise customers to provide flexible and personalized work assignments to drivers fulfilling shipping orders and rapid fulfillment of customers' brokerage needs. A key feature of Uber Freight is the ability to fill in gaps in supply chains on short notice, which requires a low reliance on fixed infrastructure and a high emphasis in developing capacity to dynamically create and alter routes.

The Uber Freight Location Tracker is a multifunction Android app that uses GPS data corresponding to a driver's location to relay related info to Uber's central systems and to the driver. Desired info includes exact info on the time of a driver's arrival and departure to and from particular facilities as represented by a perimeter against which the driver's location can be checked and info on whether a driver's shipment will be late given their current location, destination, and scheduled arrival.

Uber Freight currently has nothing in place that performs check-in/check-out detection or lateness detection automatically, which substantially undermines their ability to gather up-to-date information on how well shipments are conforming to schedules. This forces drivers to manually verify their own progress, which allows room for both human error and fraud. It also gives customers an unsatisfactory amount of info on the status of their product transfer. We have developed a product to resolve Uber's tracking-related programs by delivering an app that can perform location tracking and lateness calculations without input from the independent contractors that perform shipping work.

Functional Requirements

The final product contains four main components: The app itself, which houses the UI and tracking, the tracking component, which uses GPS services to find the user's location, the processing component, which determines whether the user is in a particular geofence, and the endpoint, which is simulated with notifications.

The app

- Has a UI that displays the destination point and an ETA, or Estimated Time of Arrival, based on the current position
- Has a processing component
- Has a tracking component
- Can interface with user

The tracking component

- Can interface with GPS services to find the device's position
- Can interface with GPS services to detect the device's presence or non-presence in a user-specified geofence
- Can feed data into processing component

The processing component

- Can interpret data from tracking component to record and prepare check-in/check-out data for transmission
- Can interpret data from tracking component to record and prepare lateness data for transmission.
- Can interface with app to provide data for transmission to endpoint
- Can store geofence data and interface with tracking component as needed.

The endpoint

- Is currently the notification interface, though info from notifications can be redirected to a central database easily.
- Can receive lateness and check-in/check-out data from the app.

Non-Functional Requirements

1. Is compatible with modern Android devices
2. Was developed with Android Studio
3. Primary program language is Java
4. All functionality must work with the app running in the background.
5. Was developed in five weeks and delivered by 6/11
6. Plans to deploy app on Firebase cut due to time constraints

System Architecture:
Data Flow

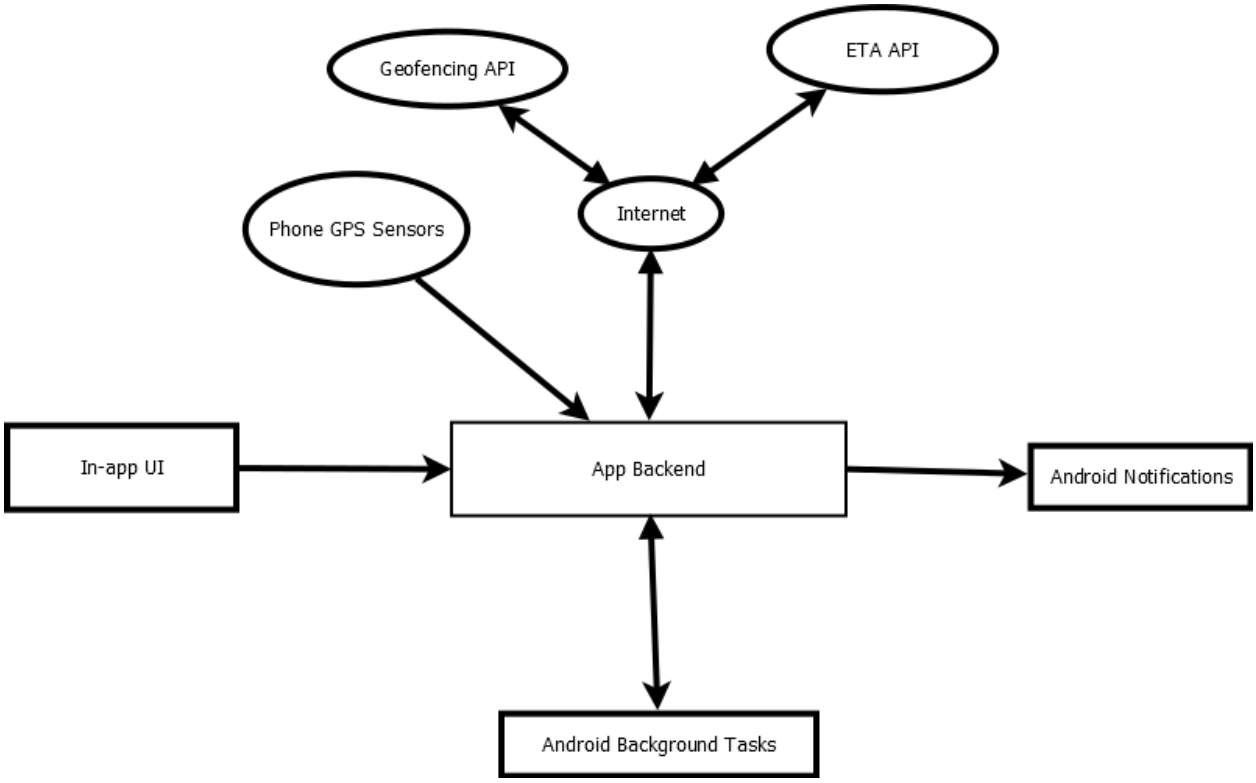

Figure 1

The app takes advantage of a number of services to perform its functions. It is currently in a proof-of-concept state, meaning that data comes directly from user input, though the code is set up such that modification to take data from another source is simple. In order to provide arrival and departure detection and lateness detection respectively, the app feeds data from the phone's GPS sensor into a geofencing API and an ETA API. This data is then fed back into the backend, which detects lateness, arrivals, and departures and constructs notification contents, which are then given to the user via the Android notification service. Because the app needs to function while in the background, the backend must be able to run as a background task in Android.

System Architecture: Execution Flow

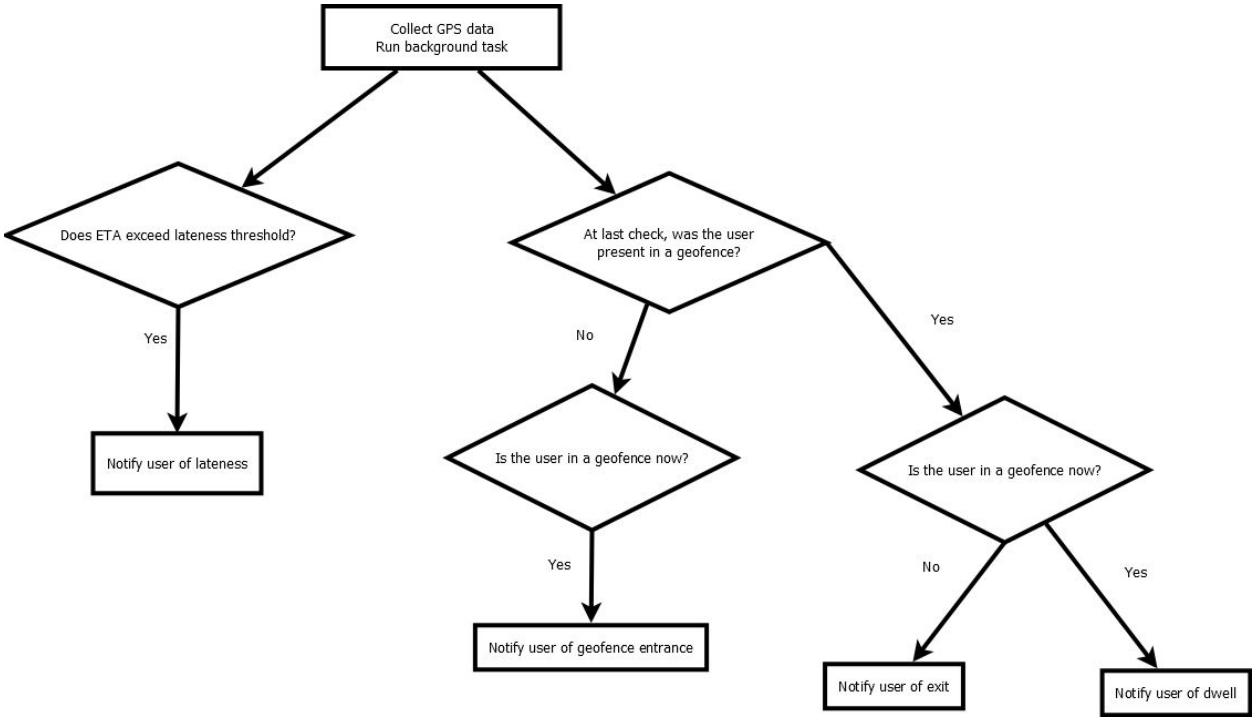

Figure 2

The app has a fairly simple execution workflow. A background task periodically collects the device's location and feeds it into an ETA API and a geofence API. If the device detects that it has entered, stayed in, or left a geofence, it sends out a notification informing the user of this fact. If the device detects that lateness is likely based on the ETA to the destination, it will notify the user that they are likely to be late.

Technical Design:

An interesting aspect of the product is the fact that it must be developed with the requirement of being able to function to a satisfactory degree in remote areas, where geolocation is substantially less precise. Modern geolocation systems use a number of complementary telemetry types to improve the accuracy of a GPS signal. While it is true that a GPS signal is available almost anywhere due to the planet-wide coverage afforded by the GPS network, relying on GPS alone gives an extremely imprecise reading of the location. As such, modern phone location services use a number of other telemetry readings to supplement the GPS signal and make location more precise. This is rarely an issue in populated areas, but location accuracy is substantially more difficult to guarantee while on the open road. A core piece of functionality desired by the client is the ability to operate in adverse signal conditions like those found in the remote locations that truck stops are often built in. The typical way to acquire the device's location automatically gathers all available GPS-enhancing telemetry, but disabling certain telemetry on the phone allows a user to gauge accuracy without a particular piece of telemetry. There is typically no reason to do this, but it allows us the ability to see which telemetry types are most helpful in acquiring the device's location and thereby predict the type and strength of compensation measures required to avoid substantial error in areas where certain telemetry types are unavailable. In order to predict the adverse impact of being unable to access certain telemetry methods, a member of the team went to an area with a strong location signal and disabled certain telemetry features. We then recorded the error in location caused by the absence of this telemetry method. All figures in the below table are approximate, as the degree of accuracy of the map used to pinpoint actual real-world location by comparison to landmarks is unknown.

Telemetry Method Disabled	Error
None	10 m
GPS	10 m
WiFi	30 m
Cell Internet Network + WiFi	50 m
All except WiFi	10 m
All except GPS	80 m

Figure 3

The data in Figure 3 provides a number of novel insights regarding location accuracy. The most important point is that WiFi-based tracking is more accurate than any single type or combination of GPS-assisted tracking, and can reliably provide highly accurate geolocation even in the absence of any and all other location methods. Therefore, it can generally be assumed that tracking will be as accurate as it can be within most truck stops, as WiFi access points can generally be expected to be present within them. Note that actually being connected to a network is not a requirement for WiFi tracking and that the mere presence of WiFi networks that connection is possible to is sufficient. That said, even the error caused by using GPS alone is unlikely to make a particularly large difference, as the only time such conditions should occur is on the open road, where only ETA needs to be calculated. If accurate location can generally be assumed within a truck stop, even hundreds of meters of error are unlikely to cause a significant difference in projected arrival time.

Another interesting aspect of the product is the need to build an app that can deal with assorted user-induced service interruptions. A driver cannot reasonably be expected to keep the app open during the entire haul, and therefore the app must perform all of its functions in the background. In order to maintain background function, system permissions to work in the background must be secured from the user. Additionally, it cannot be guaranteed that the driver will be willing or able to keep their device powered on during every ride, so a service to restart location tracking on device reboot was also necessary to maintain consistent service. Figure 4 demonstrates the app running in the background with its background presence cleared.

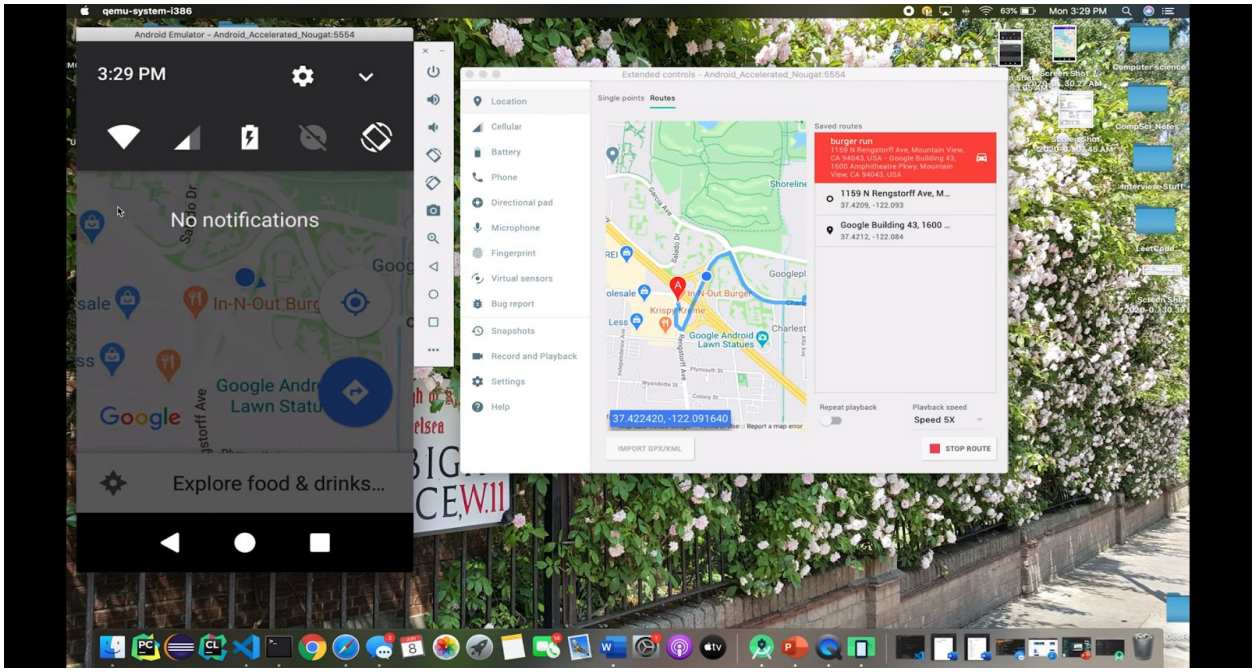

Figure 4: Video of app running while closed with its task dead

Quality Assurance

Testing Process:

- Simulation testing: Meant to remove testing error due to poor GPS readings by guaranteeing accurate readings via location spoofing on an emulator. This is meant to ensure the app can perform its basic functions at all.
    - Tested to ensure app registers geofence entrance when entering geofence
    - Tested to ensure app registers geofence exit when exiting geofence
    - Tested to ensure app does not register geofence entrances and exits when not in a geofence
    - Tested to ensure app correctly measures geofence dwell times
    - Tested to ensure app correctly reports likely lateness
- Real-World Stable Signal Testing: Meant to ensure app function in environments with stable GPS and abundant complementary telemetry to improve signal precision. This is meant to ensure app reliability in the real world.
    - Tested to ensure app promptly registers geofence entrance when device enters geofence
    - Tested to ensure app promptly registers geofence exit when device exits geofence
    - Tested to ensure app does not register entrances and exits when outside of geofence
    - Tested to ensure app correctly measures dwell times
    - Tested to ensure app correctly reports likely lateness
- Real-World Unstable Signal Testing: Meant to ensure app function in environments with unstable GPS and/or lacking complementary telemetry. There is more room for error here due to limits on the data being gathered, but measures to compensate for poor signal should be able to deliver acceptable data. This is meant to gauge app performance in unfavorable conditions.
    - Tested to ensure app promptly registers geofence entrance when device enters geofence
    - Tested to ensure app promptly registers geofence exit when device exits geofence
    - Tested to ensure app does not register entrances and exits when outside of geofence
    - Tested to ensure app correctly measures dwell times
    - Tested to ensure app correctly reports likely lateness

Results

The goal of this project was to create an Android application that would track the user's location, display it on another device, and to determine if a user has entered or left a designated area. The app would then notify the "tracker" of the ETA of the "driver". Our application has met all of our client's functional requirements, as it successfully tracks the driver, notifies the user when the geofence is entered or left, and functions even when the app is running in the background or closed. We did not implement accurate ETA's into our app, as our app would run concurrently with Google Maps which already has the feature. In addition, due to a lack of time and knowledge, we were unable to implement Firebase into our application, as such we could not have user authentication and separate user interfaces. The implementation of schedule setting, lateness notifications, and a GUI were all features that did not make it into the final product due to time constraints. Should we work on this in the future, incorporating ETA from the Google Maps API, implementing Firebase into our product, and updating our GUI would be the next tasks to work on.

Overall we learned that working remotely is difficult to overcome, but following a set schedule and frequent communication is key to overcoming that obstacle. We were able to utilize Agile methods to maintain productivity throughout the course, and frequent code reviews helped keep our program clean and running. We also learned how to implement existing API's into our project to create a new program. Initially, the project seemed much more difficult as we had very little experience in app development, and creating new location tracking code would have been very difficult. Luckily, there is existing code to help us learn the basics of Geofencing that we have implemented into our program.

Appendices:

Notes on use:
The product delivered by the team is unsuitable for direct end-user use. The app was developed with the goal of acting as a proof of concept and lacks basic functionality typically expected of a tracker that would actually be deployed to end users. Missing functionality includes an account system, the ability to receive data on new geofences from a central database, the ability to send data on lateness, arrivals, and departures to a central database, and any logging beyond that afforded by the Android notification system.

Installation Instructions:
The product has been provided both as Android Studio source code and as an apk file. To install the product, simply load the apk file onto an Android device and install it.