Open Cycling Data

Report of the Open Data and Green IT Code Camp Spring 2015

Ashraf Abdo, Dimitar Minovski, Niklas Kolbe

Lappeenranta, 08.03.2015

Table of Contents

1. Introduction

This report summarizes the activities of the Open Cycling Data group during the LUT open data code camp in spring 2015. It presents the idea of the application, the concept, and the features that were developed and how they work as well as details about the implementation.

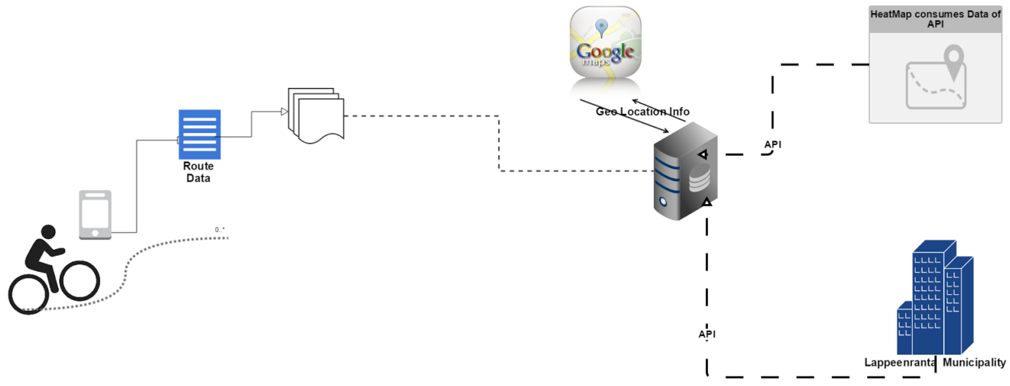
2. Idea and Motivation

The idea of the application is to crowdsource bicycling data in terms of the cycling routes and times to improve biking conditions in cities. With the help of this collected data, a city, municipality or a region could maintain and provide biking routes that correspond to the demand. Maintenance services could focus on busy routes, analysis in terms of traffic (e.g. traffic lights) could consider the bicycling data, or with the help of the data, popular destinations could be identified and new biking routes could be established.

In our personal experience as bikers, especially in Lappeenranta, we struggled with not well maintained biking tracks during winter or lacking efficient connections between certain destinations which are suitable for bikes. This motivated us to develop this application.

3. Concept

To gather the data the user just needs to run a mobile application while biking to track the route. The application allows sending the tracked routes to a web server. The server adds metadata - e.g. the name of the city, region and country of the tracked route by accessing the Google Maps Geo API- to the received route coordinates and time stamps. The data will then be stored on the server anonymously and can be accessed via an open REST API e.g. by a municipality which wants to analyse biking roads conditions. The concept is visualised in the figure below.



4. Features

The developed features during the code camp can be divided into three parts: The mobile application to collect and send the gathered data, the web server which provides the open API and an external web service using the developed API as an example of how the collected open data can be used.

4.1 Mobile App

The mobile app main functionality is to track a bike route and to send tracked data to the server. The user can review all of the recorded bike routes, and choose which of them to be uploaded to the server. In the concept of this application there is a one-click feature for the user to login and fetch the biking tracks from different external applications such as Strava, RunKeeper and Nike+. To give some additional motivation to the citizens to use the application, different information about the tracked routes can be displayed, so that the user knows where he has been and what is the distance covered during the trip. The information that could be added based on the one particular biking ride is a figure for calories burned and CO2 saved compared to a car ride.

Feature	Details	Goal	Implemented
Record Cycling Route	Tracking the GPS coordinates of the mobile phone and saving them on the phone.	Must-have	YES
Upload Tracked Data	Sending the trip information to the server.	Must-have	YES
Provide Cycling Information	Providing information to the user about the tracked routes.	As time allows	YES, but only basic tracked data
Integrate external data	Sending trips information from external sources (e.g Nike+) to the server.	Optional	Dummies

4.2 Webserver

The web server’s tasks include storing retrieved data from the mobile application and providing an open API to access the data. Before the data is stored, the web server will collect additional meta-data to the route, e.g. information about the location from the Google Maps Geo API about the retrieved route. A part of the API is to provide different response formats to ease the usage of the API from external services. These features and more details can be seen in the table below.

Feature	Details	Goal	Implemented
Query data	Querying the gathered data in terms of date and location with different granularities to make it accessible for the public.	Must-hav e	YES
Add data	Adding new routes with GPS and time information sent by the mobile application.	Must-hav e	YES
Add meta data to GPS route data	For good searches the GPS data has to be enhanced with meta information like time, country or city.	Must-hav e	YES
Different Response Formats	Providing different formats like GeoJSON or just coordinates as response.	Optional	YES

4.3 External Webservice

To show how the collected data can be used in a reasonable way, an external web service was implemented. The external web service retrieves the data from the Open Cycling API and creates a heat map which shows which cycling routes were used the most to prioritize the roads that needs to be kept in a good condition. For this, the web service made use of the Google Maps API.

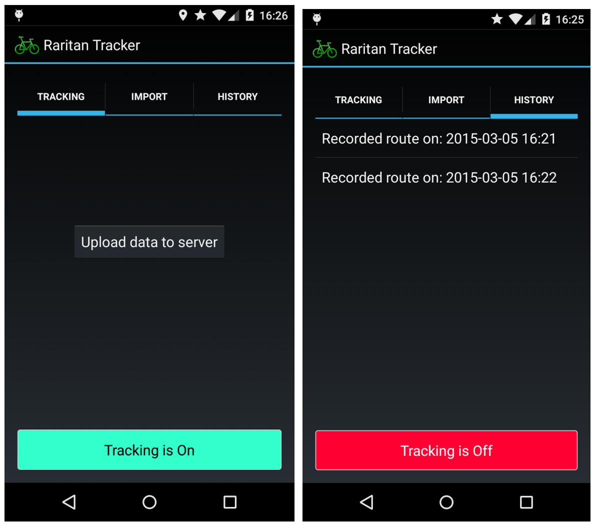
Feature	Details	Goal	Implemented
Bicycling Density Route Usage Map	A map that visualises the Open Bike Data to show busy routes.	Optional	YES
Focus map and filter data	Focus map by URL params and only request the data of the specified country/region.	Optional	YES

5. Platforms, Technologies and Implementation

The following tools and technologies were used to develop each of the solution components.

5.1 Android Application

The mobile application for Android was developed in the Android Studio IDE. In order to exploit the fully potential of the application, the phone should have a functional GPS sensor for the tracking and an Internet connection for uploading the data to the server. For testing, a phone with Android OS 5.0.1 was used.



Recorded route on: 2015-03-05 16:21

Recorded route on: 2015-03-05 16:22

The graphical interface of the application consists of three tabs, Tracking, Import and History. The actual tracking is happening in the first tab, on the click of the button "Tracking is Off" the colour and the text will be changed to "Tracking is On", and in the background a couple of things are triggered.

With the library "AlarmManager", the application is able to set a periodic asynchronous timer to wake up the class "LocationService" every 10 seconds. This means that the trigger is working in the background without interrupting other user's activities on his phone. With the implementation of couple libraries for location services, the application is able every 10 seconds to check if the user has changed his previous position and, if so, to store locally in a file the new coordinates. The timestamp of 10 seconds is chosen as an optimal period which does not use too much of the battery life, yet still provides accurate information for the trips.

The data that is stored on the user's phone is converted to JSON format, to give the possibility to the users to review their cycling trips, delete them or choose to upload them to the server. By clicking the Upload button of the Tracking tab, an asynchronous event part of the "AsyncHttpResponseHandler" library is triggered to send a POST Request to the server. This function will look into the file for a JSON object with stored cycling trips and upload them to the server.

The tab "History" also reads from the file. It looks for a JSON object with recorded trips and adds each trip as a separate item to a ListView with the exact time when the trip was recorded as a title. The user can chose to review each of the trips or delete a trip. Right now he can just review the JSON object with the coordinates of a selected particular trip, but in the next version a drawn map will be displayed.

The tab "Import" suits for importing data from different external fitness applications. The user can login to RunKeeper, Nike+ and Strava, and import the cycling trips that are completed. This feature is still not implemented, but it is planned to enable a one button click with the user's credentials to automatically import and store the data locally.

It is important to mention that the application does not store any private user’s data apart from the GPS coordinates and the time when the tracking was recorded.

5.2 Web server application

The web server application was developed in a collaborative *Cloud9* (c9.io) development environment. The web page is based on *node.js* which is a trending server runtime environment that is often used to implement scalable and efficient web services. On top of that *Express*, which is a node.js framework, was used. Express provides features for web applications, e.g. handling http requests and responses.

The data is stored in a *MongoDB*-based database which aligns well with node.js. MongoDB is a NoSQL database which supports flexible usage in terms of data structure. It is very efficient, even in huge datasets, which makes it scalable. To ease the access to the MongoDB from the node.js application, *mongoose* was integrated into the application. Mongoose provides object modeling for MongoDB in node.js. It includes for example schema modeling, typecasting and query building which makes it easy to handle the data in the application.

Basically the web service is divided into two parts: Storing new datasets, which can be found in the route *create*, and retrieving datasets, which is implemented in the route *read*.

The module for saving new bike routes to the database expects a parameter containing a JSON string. For each array element, i.e. for each sent route, following actions will be performed:

1. Accessing the *Reversed Geocoding* function of the *Google Maps API* with the first coordinates of the route. The returned location data will be attached to the route.
2. Prepare data for the database, e.g. date format
3. Store data to database

Once all routes are saved a JSON response with the status code of each route will be sent.

The module for the open API, i.e. for retrieving datasets, accepts location, time and format parameters (see chapter 6 for details). The parameters will be checked to build the conditions for the database query. E.g. the parameter *region* will be compared to various stored location values and need thus an integrated *OR* query within the *AND*, which is added for each requested parameter.

After creating the conditions, the data is queried from the database. Finally, the data will be put in the requested format (see chapter 6 for details) and a JSON response with the data will be sent to the client.

5.3 External Webservice

The web service that accesses the collected open cycling data is developed under the same circumstances like the web server application. However, it acts like an external web service. It is implemented as the default page of the web server and can be found in the *index* route. As mentioned in chapter 4.3, the idea is to display a heat map which highlights busy routes in terms of cycling. Details about the usage of this web service can be taken from chapter 7.

The external web service accepts the same location and time parameters like the API. After parsing the parameters a HTTP GET request is sent to the Open Cycling Data API. The requested format is *coordinates* since only the coordinates of the routes are required. The given parameters are sent to the API to filter the data accordingly. If a region was specified another request will be sent, but to the *GeoCode* function of the *Google Maps API*, in order to retrieve the corresponding coordinates to centralize the map to the specified region.

This data is then given to the view. The view integrates the *Google Maps Javascript API* and the coordinates are passed to the Google heat map creation which will be rendered once it the response is delivered to the client.

6. RESTful Open Cycling Data API Documentation

The complete Open Cycling Data API Documentation is available in a .json and .yaml format in the team's wiki page:

http://www.codecamp.fi/doku.php/opendata2015/group3/start#restful_open_cycling_data_api

The Open Cycling Data service can be accessed via
*https://lutcodecamp-niklaskolbe.c9.io/biketracks/{**datarequest**}/{**country**}/{**region**}?{[**parameters**]}*

- **datarequest** - defines the response format. Accepted parameters: all | geojson | coordinates. The exact format of the corresponding JSON response can be taken from the documentation or the wiki page.
- **country** - accepts a country name or country code (e.g. Finland or FI) to filter routes specified to that country.
- **region** - accepts the name of a city or area in a country (e.g. Lappeenranta) to filter routes assigned to that region.

It allows following optional **parameters**:

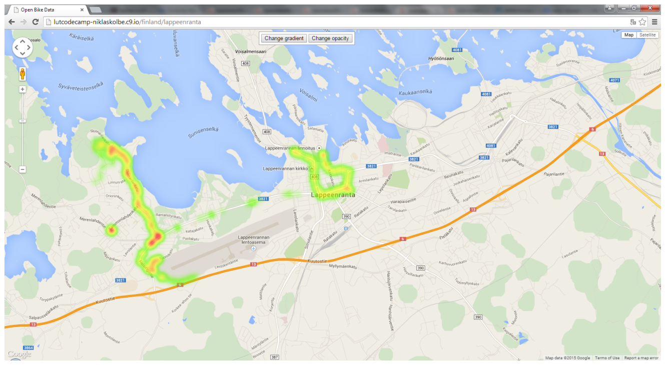
- year, month, day and limit (to limit the response to x datasets)

A valid request could look like this:
https://lutcodecamp-niklaskolbe.c9.io/biketracks/all/finland/lappeenranta?year=2015&month=3&day=5&limit=100

7. External Webservice

The external web service is accessible via
*http://lutcodecamp-niklaskolbe.c9.io/{**country**}/{**region**}?{[**parameters**]}*

The bold parameters in the URL have the same meaning and accept the same values as described in chapter 6 for the Open Cycling Data API. Country and region are optional parameters. If they are specified the cycling data will be filtered accordingly and the map will be centered to that region. This implies that the display of the heat may change depending on the specified parameters as the display depends on the underlying coordinates of the whole map.


Open Bike Data
lutcodecamp-niklaskolbe.c9.io/finland/lappeenranta
Change gradient
Change opacity
Map
Satellite
Lappeenranta