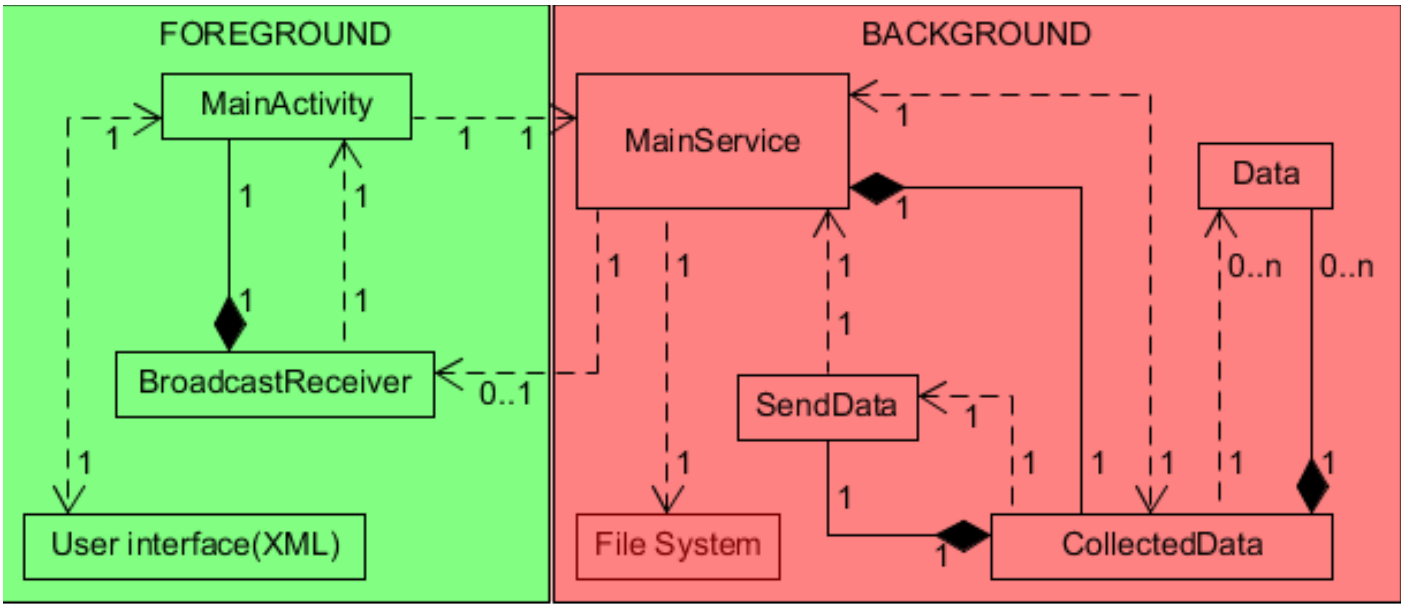
Introduction - Sofia/Patrik

Our group has developed an android application for recording and transmitting a users biking patterns to a server, which is then to be visualized on a website. We have tried to divide the work amongst ourselves by dividing it into application, server, database and website components, so we'll proceed by describing these four aspects in order as well as show a demonstration of our work. Finally, we'll introduce some of the similar work and aspects that we think could be further improved on in the future.

the mobile app - Sergej



Here a simple class overview, there we have the MainActivity class that visualizes devices activity on the google map, the MainService class which we mostly focused on, the Data class which holds the information about single data point, the CollectedData, and the SendData class which is an instance of a Runnable class. The SendData is used to create a new Thread each time we want to send the Data to our server. Notice that the CollectredData and Data classes are not the background threads and that User Interface and the File System are not classes actually, but they are included here to understand which part cooperates with wich.

A bit about the composition of our classes, the MainService creates a CollectedData variable, which in turn hold a SendData and the list of Data variables. And the MainActivity class have a BroadcastReceiver that receives broadcastmessages from our background service thread.

Dependencies of our implementation might be a bit complicated, but here is what we have. We have clearly understandable dependencies like that the CollectedData uses Data class to store all the data and uses SendData class to obviously send the data to our server or that MainActivity uses the BroadcastReceiver to receive the messages, but there is also a bit tricky dependencies like that CollectedData is responsible for starting a new Thread which will send the data over the Internet, but before we start a new Thread we want to check if the internetconnection is available in order to not waste the battery power and to avoid

unnecessary allocation of the memory space, BUT the process of checking the internetconnection has to be done in a backgound thread and the CollectedData is not a background thread, this is why we have added a static function to our MainService thread, which is used by the CollectedData class, to check the status of the internetconnectivity. In order to get the result if data was sended successfully or not, the SendData class use the appropriate static function from the MainService class.

Let's take a look at some of classes more closely.
A single Data point have a session identification number which is unique for each unique biketrip, and each biketrip usually will include more than one datapoint, it have a coordinate point which is the latitude and longitude of the map position, the time is in milliseconds (known as Unix time or Epoch time), the speed represented in meters per second and the accuracy represented in meters. The Data class have toString() function that returns a JSON string of the data.

The CollectedData class have such responsibilities as add the datapoint, get the datapoint, remove the datapoint(s), import the data from a file and prepare the data to be sent over the internet. In order to prepare the datapoints it also have toString() function, which returns the JSON string there all collected data is represented in form of JSON list.

The MainService class is responsible for tracking and handling the device's activities and in order to get devices activities we use the imported GoogleApiClient library, it is also responsible for saving the data in a filesystem when the person is biking, for sending activity information to the MainActivity class when it is up, and to clear the recorded data when the data is successfully uploaded to the server.

For information about used functions is described in the appendix.

The database and anonymity - Sofia

The database is built of one simple table. All the information collected from the phone of the user is stored in the form of the attributes:

- The session id. The session id tells us what points belong together during a so called session. This enables us to later connect the dots and calculate useful data.
- The coordinates, longitude and latitude, meaning the position of the user when the applications recorded the data.
- The time when the information was recorded
- The accuracy, which we in the future can use to figure out if the data recorded is useful or will provide us with information that will create statistically unsatisfying data.

Since the application at this point is rather small and not that much data is recorded this is a good starting point. The two things we wish to calculate for the website right now is how many bikers there were at a specific time at a specific place and at what speed bikers are traveling at different locations and time during the day. Those two facts are possible to draw from the attributes in our single table.

To ensure anonymity towards the users we have chosen to label every coordinate recorded with a session id. If we are to be able to calculate data from the database in its' current form

we need to be able to connect the coordinates of specific bikers. The session id takes away the identifying factor, like the name or username of a user and replaces it with the session id.

A session id is generated for every “session”. A session starts when a user starts biking and ends when a user has not been biking for 10 minutes. This means that the information in the database is lumped together with the session id making it possible to connect coordinates without knowing the person who made that route. The session ids are not tied to a specific user but they are generated when a session begins. The users anonymity and privacy is thereby enforces since there is no way to connect a person to a session id. And since a person can’t be connected with the data no behavioral conclusions can be drawn about a person.

Another feature that helps anonymity is a side effect of the activity update. The activity update helps us determine if a user is actually biking or just walking or driving a car. Since the sessions starts when the activity is updated to “biking” and that update is made every 30 seconds it will not be possible to determine exactly from where a biker started in reality. This makes it a bit harder so draw any behavioral conclusions about the user if data should be leaked.

the server - Patrik

RESTFUL web service
privacy/anonymization

We set up a restful java server (REST standing for representational state transfer, which is a method for building scalable client-server web services.) The clients, upon detecting wi-fi access, simply sends JSON data to the server using the standard HTTP post method.

In case anyone is unfamiliar with JSON, it is a standard for formatting human-readable data for transmission.

The server is quite simple, and doesn't do much processing at the moment, however it's meant to be able to scale with an arbitrary number of users.
We explored a few ideas for doing more advanced things like pre-processing data server-side and populating the database in a more "rich" way, for instance by fitting datapoints to actual streets and bikepaths, which might allow for more interesting (and cheaper) queries, but in the end we didn't pursue those options due to time constraints. For the future, as the website is expanded with more features, that would certainly be interesting.

We also had to consider anonymization and making sure that the application was as non-intrusive as possible. As mentioned, no one should be able to identify a user based paths in the database. We don't actually store anything related to any one particular user on the server or in the database. The paths are split up into smaller sequences so that the full path (which could, for instance, be someone travelling from home to work) isn't represented as such in the database. Even if someone was to find such a path, they wouldn't be able to derive any other paths taken by the same user as we don't store anything related to individual users in the DB.

demo/the web site - Zijian

Hello, everyone. I am going to present how do we display the data which are fetched from mobile phone users. In order to show the pattern of biking paths, we will have a website. In the first

place, we also considered how are we going to display the pattern to a government officer or a corporate user. Shall we just display on the mobile app? No, the screen is quite small so it is hard to analyze. Shall we develop an app for different platform to get the data from the server and create the photo on the platform? I think you can guess the answer from my question. If we want to develop an app, we need to create at least three versions: a windows version, a Linux version and a Mac version. Therefore, that is very time-consuming and money-consuming. So what did we decide to demonstrate at the end is that we are going to display on a website. It is obvious that all different systems can scan the website even Android and IOS. Then our data can be checked by many different users at the same time together.

OK, next let's look at the website. Firstly, it would be the page of introduction. The content of this page has been talked by my partners. Secondly, it would be the page of bicycling route. The bicycling route shows that which roads or trails are friendly to bikers. You can see there are three different types of green paths. Each of them indicates different levels of friendly to biking. Dark green indicates a dedicated bike-only trail, for example... Light green indicates a dedicated bike lane along a road, for instance.... And dotted green indicates roads without bike lanes, but are more appropriate for biking, based on factors such as traffic or the width of the road. This helps cyclists better plan their routes, we embedded it in our website is used for comparison with heatmap. In order to understand better the meaning of this bicycling route for our users, let's go to heatmap first, and I will continue to talk bicycling route when you got what is showing on the heat map.

In order to better understand what is a heatmap, let`s go to see a sample in the official website of Google map API. This is the heatmap of Los Angel. A heatmap is a visualization used to depict the intensity of data at geographical points. When the Heatmap Layer is enabled, a colored overlay will appear on top of the map. By default, areas of higher intensity will be colored red, and areas of lower intensity will appear green. in a summary, the heatmap is likely showing the temperatures, like the red area, you can feel in there the temperatures much be very high which means that there are more cyclists in the area. So this is the way how are we going to exhibit t

This create_heatmap.php page is not simple as you have seen. Let look into it, when we select a date, it displays the following date, actually, all the following dates are queried from the database, which means that on the date, there is at least a point. next the time is easier, we just put the time range in here. Then, you can see the note cannot be blank in both options, since there will be too much data for the heatmap, it may take very long time to display the content. So we added an alert function. If you did not select any of the two options, there is an alert window will show up to recommend you to select an option at least. So what happens if there is no any available data in your selected range, let's try it. There is also an alert window tell you "sorry, there is no data during your selected range".

This is the pattern of cyclist on the date at Uppsala. You can

And you can see that there are four function buttons. Let me introduce you the functions. By the button of toggle heatmap, you can hide the heatmap markers. By the button of change gradient, you can change the intensity of marker’s colour from green to blue, which makes the heatmap marker clearer. Opacity could change the level of transparency applied to heatmap.

Now, you understand what is heatmap, we could come back to the bicycling route. The main function of this page is allowing users to compare heatmap and the bicycling route, then the user could find if the road used most frequently is a cyclists friendly road, is that safe and convenient for the cyclists.

So far, you have seen all functions we have on the website, but have you noticed that we have the searching box and the map is a world map. So our application and this website can be used in anywhere around the world, not only in Uppsala.

other simular projects (check Aleksanders links) - Patrik

We did find a masters thesis that was published earlier this month, that dealt with something fairly similar to our project. In this thesis, bike data was also collected, but rather than having actual users with apps installed on their phones,
she developed an app for evaluators to count bicyclists within a given area and submit their counts via the android application (just to simplify the task as opposed to using pen and paper). The main goal of this thesis was to use bike data in conjunction with vehicle data and data from
actual collisions involving bicycles, in order to better understand and prevent accidents. The upside of our way of collecting data is obviously that as long as we can provide some incentive to use the application, there is no need
for evaluators to do the work of counting bicyclists.

hardships

improvements/future developments - Sofia

First of all the application structure could be somewhat simplified. To clean up the class diagram we could move the sending of data to the main service.

We could also implement security in any form since security has not been the focus of this project.

The applications in the phone could be expanded to show the user individual information about biking habits as many other activity tracking applications does now a day. Another feature would be to implement the same functions we have on the website to be available on the phone.

To ensure that the data we collect will not result in statistical errors we could use the accuracy prediction to remove or reconstruct data points that seem unlikely. For example, when information was recorded about me biking to school some coordinates was recorded in Fyrisån. This particular example is not a big problem since it was a miss of at most 10-15 meters. But it is worth thinking about how we are to get as accurate information as possible. This calculation would maybe best be made by the server so that the users' phone doesn't have to make unnecessary calculation.

Improvements to the database:

The database is right now quite small. It will have to be further developed if it is to hold the amount of data the bikers of Uppsala will generate. It might be a good idea to compute statistically interesting data from the larger dataset and store it in a more compressed way in other tables to make future querying more efficient.

To increase anonymity and privacy the data could be reorganized in some additional way that doesn't even require the session ids. That would entirely eliminate the possibility to

recreate routes that could be used with approximate knowledge of users' habits to learn more about the users.
If we were to make the applications more personalized we would have to expand the database to hold information about specific users. In that implementation we would still have the problem of the users' privacy and would have to come up with a way to store user specific information without compromising the users privacy.

The website:
To make the website a bit more user friendly we could make the time range a bit more flexible. Right now it is possible to select a day and an hour during that day and we would display data for that time. We would like to add more detail and enable the user to select down to the minute what time he or she wishes to examine. To make the site even more useful it would be nice to implement a slidebar so that it is not just possible to examine one moment in time but rather time spans with a start and a stop time.

It could also be nice to be able to save maps to make it easier to compare results from different searches.

The server:
The server should be extended with better security (encryption, authentication, proper logging etc). As mentioned, in the continuation of this project it will probably be wise to let the server process data and populate additional tables in order to allow for fast queries and additional features on the website.

Appendix

```
class MainService:

    public static void successfullyUploaded()
    // Sends a broadcast Intent (with action 'DATA_UPLOADED') to the
    // MainActivity, clears the file with the collected data and resets current
    // CollectedData object.
    // **This function is used by SendData thread, to verify that the data was
    // uploaded successfully.

    public static void failedToUpload()
    // Sends a broadcast Intent (with action 'UPLOAD_ERROR') to the MainActivity
    // **This function is used by SendData thread, to verify that it was unable to
    // upload the data.

    private boolean clearCollectedFile()
    // Writes an empty string in a file, and the path to the file target is stored in a
    // private string named filePath. By default 'sdcard/collected_data.txt'.

    public static boolean isOnline(Context context)
```

```
// Determines if the device is connected to an external router or if the cellular
// internet is enabled.
// **This function is used by the CollectedData object. In order to start new
// SendData thread, the returned value if this function has to be ‘true’.

private boolean addUpdate(String text)
// Append the string content to the file content, referred by the filePath string.
// The returned boolean determines if the appending process was successful or
// not.

protected synchronized void buildGoogleApiClient()
// Build an appropriate GoogleApiClient and start the connection process.
// Uses only in onCreate().

@Override
public void onCreate()
// Execute the function named buildGoogleApiClient.

private CollectedData data()
// Return the pointer to the CollectedData object, that contains all the collected
// data.

@Override
public void onStart(Intent intent, int startId)
// Defines static variables (boolean named active and ) on service start.

@Override
public void onConnected(Bundle p1)
// This function is called after GoogleApiClient has successifully connected to
// google services. It initialize all variables, that is needed for future
// update/remove function calls. Default input ‘p1’ is never used.

private void requestRecognitionUpdates(int sec)
// Start the Activity Recognition updates by using ActivityRecognitionApi and
// current GoogleApiClient. Integer sec is the intermediate time between each
// activity recognition Intent.

private void requestLocationUpdates()
// Start the Location updates by using FusedLocationApi and current
// GoogleApiClient.

private void removeRecognitionUpdates()
// Stop the Activity Recognition updates by using ActivityRecognitionApi
// and current GoogleApiClient.

private void removeLocationUpdates()
// Stop the Location updates by using FusedLocationApi and current
// GoogleApiClient.
```

```
        private long longDiff(long l1, long l2)
        // Return the difference between l1 and l2.

        @Override
        public void onLocationChanged(Location p1)
        // If the boolean named record is true, then it uses the information from input p1
        // in order to calculate the speed, current session id, and save the results and
        // the additional information from p1 to the file (referred by filePath) and to the
        // CollectedData object.

        private String getActivityName(int type)
        // Integer type is compared with static types in DetectedActivity class, and the
        // appropriate string is assigned to the return value.

        @Override
        public int onStartCommand(Intent intent, int flags, int startId)
        // If the incoming intent has the Acivity Recognition Result, it send a broadcast
        // intent that include necessary information from the incoming intent's extracted
        // data. Further actions depends on the collected type of activity and a boolean
        // named 'record'. In general it manages the activity of location and recognition
        // updates.
        // If the action of incoming intent is 'SERVICE_STOP', then onDestroy function
        // is used.



class MainActivity:

        @Override
        protected void onCreate(Bundle savedInstanceState)
        // When activity starts it send a start service Intent in order to start MainService
        // (if it is not 'active'), define a BroadcastReceiver in order to get broadcast
        // intents, define other needed variables.

        @Override
        protected void onResume()
        // Register the BroadcastReceiver and prepare the application's map interface
        // before showing it to the user by using a function named setUpMapIfNeeded.

        @Override
        protected void onPause()
        // Unregister the BroadcastReceiver.

        public void updateActivity(String name, int confidence)
        // Updates the title of the marker on the application's map. Is used by the
        // BroadcastReceiver and function named changePosition.
```

```
    private void setUpMapIfNeeded()
    // Check if map is already defined, if not create a map with help of a function
    // named setUpMap.

    private void setUpMap()
    // Applies needed settings to the GoogleMap object.

    private void updateMarkerTitle(String title)
    // Updates the title of marker for user's position with the input string named title.
    // This function is used in function named updateActivity.

    public void changePosition(double latitude, double longitude)
    // Update marker's position and title on the map and update the coordinate
    // values in the titlebar of application's interface.
    // This function is used by the location change listener, defined in function
    // named setUpMap.



class CollectedData:

    public boolean uploading()
    // Return a boolean value which tell about if the SendData background three is
    // currently running or not.

    public boolean dataIsEmpty()
    // Will return true if the size of ArrayList, used to hold data, is zero, false
    // otherwise.

    public void add
    (int id, double latitude, double longitude, long time, float speed, float accuracy)
    // Create new Data point with function's input as values and add it to the main
    // ArrayList of data.

    public boolean isOnline()
    // Return the value of the static function in MainService class, named isOnline.

    public void resetData()
    // Remove all data from the ArrayList, by using the function named
    // removeLastData.

    public int getLastSID()
    // Return the session ID of last element in the ArrayList.

    public long getLastTime()
    // Return the time value of last element in the ArrayList.
```

```
        public boolean uploadData()
        // Manages the start of SendData thread and returns a boolean representing if
        // thread was started or not.

        private void removeLastData(int last)
        // Integer last defines the number of last elements in the ArrayList, those
        // elements are removed at the end of function's execution.

        public boolean importFile(String path)
        // Read the file referred by input string path and fill in the ArrayList if the
        // structure of context does match requirements.

        public String toString()
        // Return a current class definition, including the ArrayList, in form of JSON
        // string.

        public String dataToString()
        // Return ArrayList in form of JSON string.

        public void finalization()
        // Reset data and call default finalize function.



class Data:

        public int getSID()
        // Return the stored session ID.

        public void setSID(int sid)
        // Change value for session id to the input integer sid.

        public double getLatitude()
        // Return value of latitude.

        public double getLongitude()
        // Return value of longitude.

        public long getTime()
        // Return value of time.

        public float getSpeed()
        // Return value of stored speed number.

        public void setSpeed(float speed)
        // Assign the value of input float called speed, to the current speed variable.

        public float getAccuracy()
```

```
        // Return the value of accuracy.

        @Override
        public String toString()
        // Return all stored information in form of JSON string.

        public boolean finalization()
        // Run finalization process.



class SendData:

        public void setInput(String i)
        // Assign the variable string that will be sended, with the value of input string
        // named i.

        private void setUploading(boolean uploading)
        // Assigns the boolean that indicates if the thread is running by the value of the
        // input boolean named uploading.

        public boolean uploading()
        // Return the boolean which indicate if the thread is running or not.

        public void run()
        // Setup the connection with the server and upload the defined string.
        // This function is being executed when thread is running.
```