Project Report

Ubiquitous and Mobile Computing - 2015/16

Course: MEIC

Campus: Alameda

Group: 14

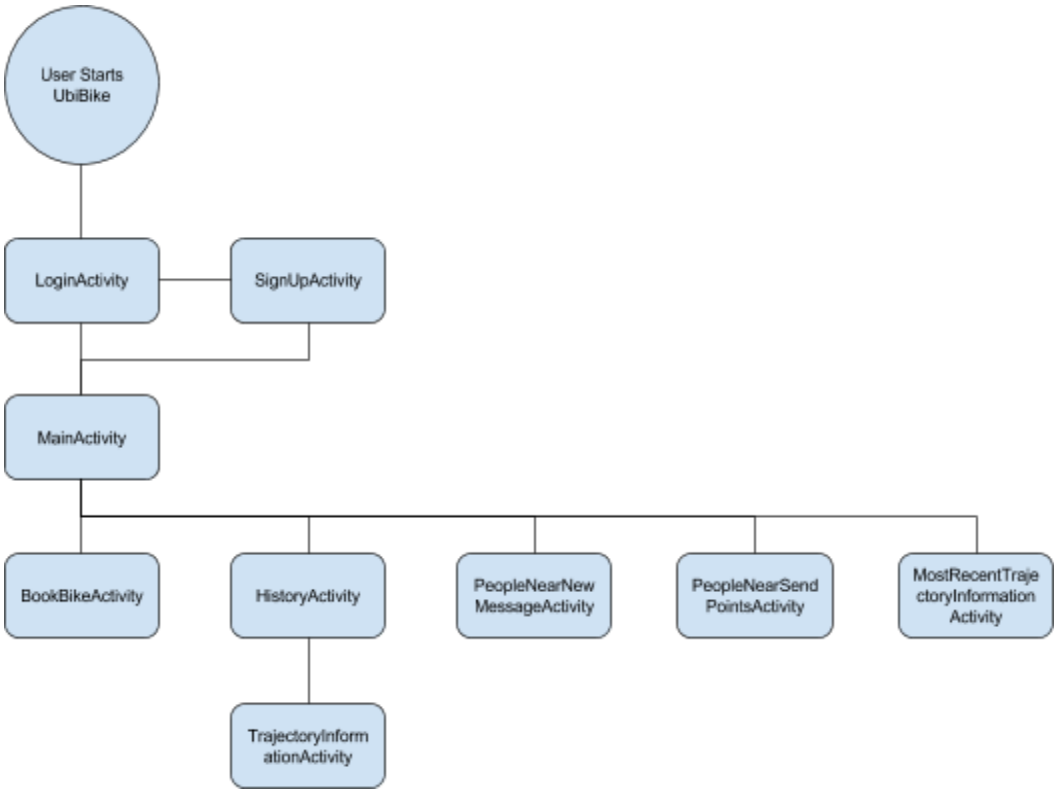
Name: Ricardo Abreu Number: 76370 E-mail: rjlabreu@gmail.com

Name: Luís Gomes Number: 72904 E-mail: luis.ribeiro.gomes@tecnico.pt

1. Achievements.

We considered to have fully implemented all the desired features, both the baseline and advanced ones. The proposed table was hidden to save some space in this final report.

2. Mobile Interface Design



The UbiBike starts at Login. From there, the user can sign up for the first time, going to Sign Up activity, or fill the username and password field and send it to the central server to login himself if the user has already signed up previously, receiving his user information before moving on into Main activity. The Sign Up activity adopts the same behaviour; it differs from Login activity because the server will sign up the user if it signing up for the first time. Both activities require mobile or wifi network to communicate with the central server.

On the Main activity, the user will have at disposal several buttons, each one for a different functionality: book a bike (requires internet); see his trajectory history (requires internet); send messages and points to nearby UbiBike users; watch most recent trajectory on a map layout; and logout application. A service will be initialized here to track user location and detect nearby UbiBike users and beacons.

On Book Bike activity, a map will provide several markers, each one representing a station. When the user clicks a station's marker, a pop up will ask the user to book a bike from the selected station. After that, the user either can enter the station and grab the assigned bike or unbook the bike or book another bike from another station, unbooking the current bike. While the user has a booked bike, the map will draw a line between user current position and the station with user's booked bike.

On History activity, there will be a list of all trajectories the user has made since it's sign up. When the user clicks a trajectory, a map appears - the Trajectory Information activity - showing the selected trajectory. The Most recent Trajectory Information activity acts the same way as Trajectory Information activity, but it only shows the most recent trajectory.

Both People Near Activities have a list of all the current peers that are currently in the user's range and a button for the user to explicitly request an update to this list. The user's current points and the peer he is connected to are also displayed.

3. Baseline Architecture

3.1 Data Structures Maintained by Server and Client

The server stores three lists: one for the users, one for the stations and one for the traded points. Each station has a list of bikes and each user has a list of trajectories.

At the client it is stored a given user's past trajectories and message history. We also store a map which maps virtual ip's to usernames. This is used when dealing with points or message exchanges. These data structures are not stored persistently and such are updated on each login.

The client and the server rely on JSON objects for communication.

3.2 Protocol for Bike Pick Up

When a GPS event is detected, we match it against all the station's coordinates to check if the user entered a station. After entering a station we wait until the bike we reserved is near us. We do this by matching every peer name in range against the beacon identifier that was assigned to us when booking the bike. After being in range of our beacon (bike) we wait until the user leaves the station with the bike to consider the pick up complete. Once more, GPS and WifiDirect events are used to calculate the user's location and detect the beacon proximity, respectively. We consider that a user is outside a station when his coordinates and a given station's coordinates differ in at least 20 meters.

3.3 Protocol for Trajectory Tracking

After picking up a bike, the application starts to accumulate locations from gps where the user stands until the smartphone stops sensing the beacon, emulated by WifiDirect. The trajectory created is sent to the central server if the mobile or wifi network are enabled and available; if not, the trajectory is stored until there is network available and enabled. If the user is outside of any station, the user needs to re approach its bike and the smartphone needs to sense the emulated beacon to start a new trajectory; otherwise, the bike is considered dropped off and the user needs to book a new bike and pick it up to start a new trajectory.

3.4 Protocol for Bike Drop Off

We consider that a bike was dropped off in two situations: the user was riding a bike and stopped sensing the beacon, or, a user was riding a bike and stopped sensing the beacon but in a station's coordinates. The difference between these two events is the location where the user stops sensing his bike. Whenever a user stops sensing his bike, we upload (if possible) the trajectory until that given point to the server.

In the first scenario, we assume that it will still be possible that the user might sense his bike again in the future - maybe he just stopped riding his bike to enter a shop along the way. In the second case, we consider that the bike was delivered to our station finishing that user's usage of that bike. If the user wishes to ride it again he must book it.

3.5 Protocols for Point Sharing between Bikers

Whenever a "network membership changed" event is caught by our service, if we don't already know all the usernames associated with every member in our group - we store the (virtualIP ; username) pair - we open a socket connection to that/those unknown virtual IP's so that they can answer back with their respective usernames.

The user is presented with a list of peers in his range (note that some of these peers might not be in his group) from which he selects the one he wishes to send points to. After sending the points to his peer, the sender issues a request to the server so that the server can keep track of all the points traded between users and validate the transaction (see Security and Robustness sections). If no internet is available at the time, the request will be sent whenever the user is back online.

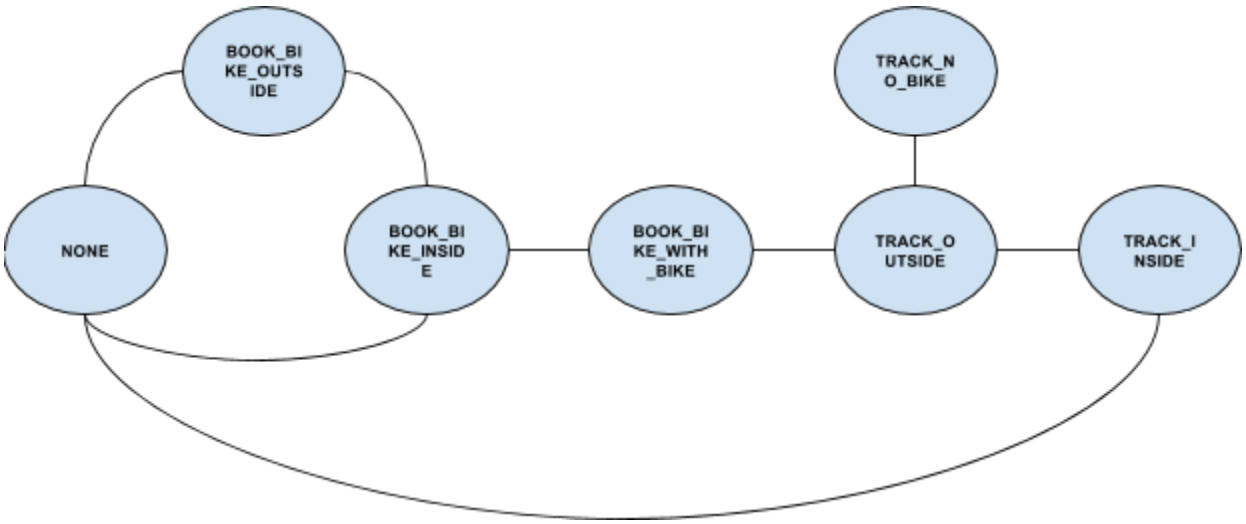
When points are received, if the receiver has internet connection, he sends a request to the server to check if the points he currently has on his cell phone are consistent with the ones he has on the server. If he has no internet connection, this checking will be done later when he is back online.

3.6 Protocols for Message Exchange between Bikers

Whenever a “network membership changed” event is caught by our service, if we don’t already know all the usernames associated with every member in our group - we store the (virtualIP ; username) pair - we open a socket connection to that/those unknown virtual IP’s so that they can answer back with their respective usernames.
The user is presented with a list of peers in his range (note that some of these peers might not be in his group) from which he selects the one he wishes to send messages to. After it he writes and sends his message which will be sent through socket connections

3.7 Other Relevant Design Features

After logging in, besides receiving the user’s current score and some other relevant information, we also receive (if it exists) the user’s most recent trajectory. This was implemented so that a user only needs internet connection when logging in, booking a bike or if he wants to see more than his last trajectory.
In order to orchestrate GPS and Wifi Direct events we chose to run a service that implements both the GPS and Wifi Directs callbacks. This enabled us to follow a state machine approach to deal with both the bike pick up and the bike drop off events - figure XX.



We considered that it should only be possible to pick up a bike if it was previously booked. This way, after a bike is booked the service enters the BOOK_BIKE_OUTSIDE state. When the user enters some station’s location we change the state to BOOK_BIKE_INSIDE. We do this by checking the user’s current location against every station’s location. Whenever we detect that we are near our bike we enter BOOK_BIKE_WITH_BIKE state. This is done by matching the beacon’s ID - that was assigned to us when the bike was booked - against all the device’s name that are currently in our range. After the user leaves the station (user’s distance to station >20 meters) we consider that his new trajectory has just begun - TRACK_OUTSIDE. From now on we register the user’s trajectory according to the GPS’ coordinates received. A new trajectory is sent to the server whenever we detect that the our beacon’s ID is no longer in the in range peers’ list. See Section 3.4 for more information on this protocol.

4. Advanced Features

4.1 Security

As we considered the user’s cell phone as a secure environment we focused our attention to the network threats. We aimed to provide both message integrity and confidentiality between the client and the server communication when dealing with traded points requests.
This way, upon sign up the server generates a symmetric secret key for the user and sends it back to him so that it can be stored in the user’s mobile phone. Unfortunately, this exchange is done insecurely - in section 6 we present a solution to this problem.

Before sending the request to the server containing the relevant information about a points transaction the client signs it. He does this by applying a SHA-256 hash on the request and then ciphering it using AES and his previously shared secret key. We chose a symmetric cipher because it is less resource intensive and still provides the desired properties.

4.2 Robustness

The server always knows how a certain user gained his points because whenever users trade points the sender of the points, always sends a request to the server informing it of the transaction he just performed. This transaction is securely sent through the network - section 4.1. If the sender doesn't have internet connection this transaction will be saved and later sent when he is back online. This way the receiver can see points on his screen that are not yet consistent with those he has on the server. Whenever both the sender and the receiver are back online their points will be synchronized.

5. Implementation

The central server was implemented in java 8, thus being cross-platform. It also uses a json library for client-server communication.

The request to/from the server were made in JSON. On each key, instead of giving a human-friendly but long name (eg. "trajectory"), each field is represented by a number with a length of 4-bytes. This number come from a finite enumerable common to both processes.

The UbiBike implements all activities from the activity wireframe. Besides them, for error, network, gps and map event handling, abstract activities were made to bind and unbind extended activities with event managers and define default UI behaviours for each event (eg. show "No Internet" alert when the internet was disabled): respectively, the ErrorHandlerActivity; the NetworkHandlerActivity; and the GpsAndMapHandlerActivity.

There is only one service on UbiBike - the UbiBikeService -, which manages all book bike, gps and wifi direct related events to detect the user current situation. It implements a state machine, using an enumerable UbiBikeServiceState and several state transition events.

Except from between LoginActivity and MainActivity and between SignUpActivity and MainActivity, every activities communicate with each other by using intents when starting or ending a certain activity. There are several ways of communication between UbiBikeService and the activities: with MainActivity, PeopleNearSendPointsActivity and PeopleNearNewMessageActivity, the service references them directly; with BookBikeActivity, the activity sends a broadcast to the service to notify if a bike was booked or unbooked.

The user information and UbiBikeServiceState are stored in a Application Context called UbiBikeApplication. The user information is added after the server authenticates the user and replies it with user information and list of stations, and is accessed by the MainActivity and UbiServerService. The UbiBikeServiceState is used by UbiServerService for event management and by the BookBikeActivity to prevent booking bikes if the user has booked or is using a bike.

Besides the main application thread, there are two kind of secondary threads: the NetworkThread and the ClientThread. The first one is used by the NetworkHandlerActivity to detect if the network was enabled. The second one sends requests and receives replies from central server.

The clients and the central server communicate between themselves via socket connection. Both requests and replies are send in json and both the server and the client handle each message differently according to the type of message, defined using the shared classes UserReplyType and UseRequestType.

We chose only to store a user's secret key in the mobile's internal memory. In section 6 we propose a solution to the past trajectories storage.

Since we consider to have implemented all requisites fully, we describe some optimizations in the following section.

6. Limitations/Optimizations

In future versions we would like to improve some aspects regarding resource consumption, scalability and security.

Firstly, the key exchange protocol that happens on sign up is clearly weak and non practical. In a future version we could change it to a Diffie-Hellman key exchange protocol.

As of now, when we have trajectory and traded points history to send to the server (because we were offline) we make a request for each trajectory/traded point. If we have N trajectories/traded points in the queue we have to send N requests through the

network, this is obviously non scalable. This way, we would like to aggregate history events in a single request despite them being trajectories or traded points requests.
Besides that, when we fetch a given user's past trajectories, we fetch them all. This is again non-scalable. One improvement we would like to implement is to limit the past trajectories sent from the server in these situations. One alternative would be to only send the past 10 trajectories so that the user can still see some of his riding history. Then, on the history activity we would insert a "see more" button that when clicked would fetch 10 more trajectories. This process could be repeated until there are no more past trajectories for that user.
Finally, we could save the user's past trajectories to the mobile phone's internal memory to avoid having to fetch them in every session. This would save request in certain situations, but we would have to be careful to also implement some version control mechanism to deal with inconsistencies.

7. Conclusions

After developing this system we can now better understand the challenges about developing applications in a mobile environment. We faced several problems from context-awareness when dealing with bike pick up and drop off events to security ones when trading points.
Our worked focused on location tracking and Wifi Direct communication.
As seen in section 6 we couldn't implement every feature we had in mind. Some issues like battery usage, data consistency and server replication could be tackled in future versions.
The practical part of this course could improve by showing some "design patterns" or good practices specific to the mobile architecture. Also shed some light on how to test this kind of applications - android testing support library.