Smart Bracelet 101

Developer Guide

Anton Peniaziev

Alon Vizel

Sapir Kremer

Last updated 30.06.2017

Table of Contents

Introduction 3

About the Project 3

Smart Bracelet 101 3

Bluetooth 4

General Description 4

Connection Management 5

Sending Data to Bracelet 5

Next Steps and Enhancements 5

Data Classes 6

Tent, Patient, Bracelet and Treatment 6

Tent Activity 7

Logger 7

Next Steps and Enhancements 7

Web 8

General Description 8

TreatmentsTable 8

Login and Logout 8

Location 8

Evacuation 8

Sending Data to Web 9

Next Steps and Enhancements 9

User Interface 10

Login Activity 10

Patient Information Activity 11

Next Steps and Enhancements 12

Introduction

About the Project

The project smart bracelet was made by students of computer science department at Technion, Haifa as a part of project in software. The main goal is to assist Israeli Medical corps in the process of registration and condition monitoring of wounded soldiers on the battlefield. The conservative approach to record immediate treatments that were performed on the patient was to write them on the patient's body or attach a piece of paper. When information on initial procedures is critical for patient's survival, the usual approach is full of drawbacks and human factor that is crucial data could contain mistakes or even lost during patient transportation.

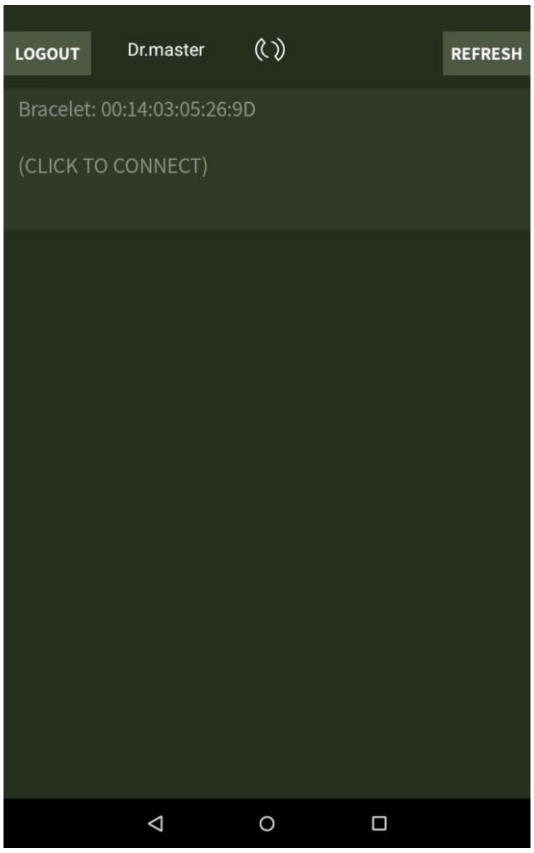
The smart bracelet system comes to reduce the human factor while recording first aid treatments. One domain of the system is wrist bracelet based on Arduino, it contains NFC scanner and Bluetooth. Paramedic needs to scan all treatment items used with this bracelet, it is assumed that all standard first aid items will have an NFC sticker attached. The second part of the project is android application that is described in detail in this document. It receives the information from number of wrist bracelets simultaneously, provides the list of treatments performed for each patient in the Bluetooth reception range. The application user can also add or edit information. All this data is being continuously synchronized with remote servers. Here comes the third part – web service. The web page allows to get statistical information about all treatments performed anywhere and perform queries on database.

Smart Bracelet 101

The Android application was developed as Android SDK project and supports API 22. All layouts were tested only on Nexus 7 screen. Described software could be logically divided into four main parts: Bluetooth, web, data update logic and user interface.

Bluetooth

General Description



Bracelet: 00:14:03:05:26:9D

(CLICK TO CONNECT)

All communication between bracelets and Android device is performed via Bluetooth. After user enters valid credentials in the login screen Android Bluetooth device starts scanning for nearby devices with familiar names which are hardcoded (“HC-05”, “HC-06”, “11”, “gun1”).

Note: those are just names of the devices that were available in the Technion lab, in order to avoid displaying unsupported devices bracelet Bluetooth should be given a name that unlikely collides with any other device’s name. However HC-05 and HC-06 are also model names of serial Bluetooth devises from Arduino kit.

Main logic for managing scanning, implementation of functions and data structures for serving the user code are located in the **BTSsevice** class that implements BTservice interface. All scanned devices are contained in BTService private field data structure

private ArrayList<BluetoothDevice> _discoveredDevices;

This data structure is exposed to user code by the function **getDisconnectedListsMap()** . The result is displayed by a listview:

Each row is clickable and the click implies attempt to pair to a device and then to connect to it. If devices are already paired only connection attempt will be performed.

Both actions above are filtered and handled by

private final BroadcastReceiver mReceiver

It has two intent filters:
ACTION_FOUND – which is activated when a new device scanned. If it’s a known device it will be added to **discoveredDevices.**

ACTION_PAIRING_REQUEST – if devices are not paired a password dialog will be prompted immediately, but the process is handled programmatically and the known password which is 1234 in our case will be provided and the devices will be paired.

Connection Management

The next step is performing a connection. The function
public void connectByMac(String mac)

is called . It initiates creation of the next important class field – **SerialBTConnector** , its main goal is to open a socket with specified Bluetooth and to initiate an Rfcomm connection. Then it passes control to **ConnectionManager,** this class is actually a per-connection thread.

It overrides a run() function of **Thread** class. The main task is to manage input and output streams of given socket. ConnectionManager also has an access to main BTservice data structure

private ConcurrentHashMap<String, List<String>> _macToReceivedBraceletData;

which is a mapping MAC address => list of received messages.

It's clear that ConnectionManager needs somehow to understand where message starts and ends since data of macToReceivedBraceletData hash is partitioned in granularity of single message. To define it uses function **handleBraceletMessage** which has some hardcoded symbols defined in the bracelet spec.

Sending Data to Bracelet

ConnectionManager thread provides a function for writing to socket it manages

void writeString(String str)

which's functionality is exposed to user code by wrapper from BTservice

public void addDataToBeSentByMac(String mac, String data)

Another important data structure is

private LinkedList<String> _onConnectionBroadcastList;

which stores list of messages that needs to be sent to bracelet immediately after the connection is established. In current implementation it contains only user's id, which is composed in message of the form <1,8234567> where 1 is the message type defined by wrist bracelet team and 8234567 is doctor's id.

Next Steps and Enhancements

Further developer should consider separating all protocol-related logic into generic Adapter. Also all technology-related logic i.e. serial Bluetooth connector could be implemented as adapter to support another technologies, for example another android Bluetooth devices.

Also the limitation on number of simultaneously connected bracelets, which is 8 on most Android platforms could be solved by implementing some kind of round robin algorithm that will disconnect/connect to surrounding devices alternately.

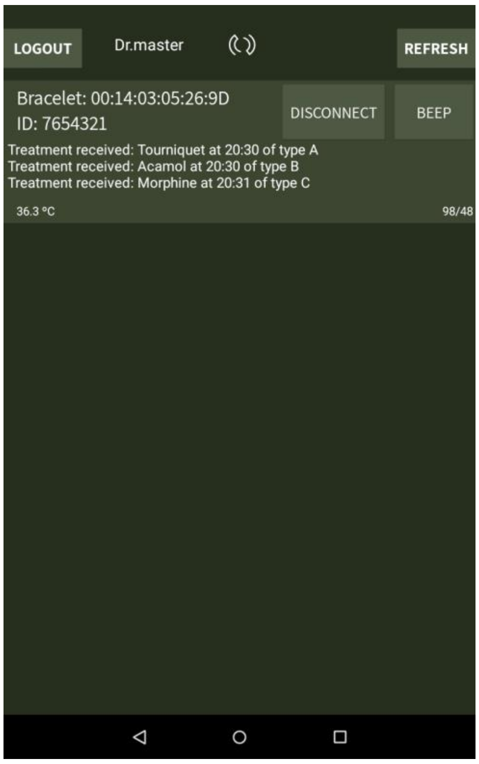
Data Classes

Tent, Patient, Bracelet and Treatment

The main data which we are interested to store in our application is list of treatments and to whom they were given.

Tent class contains **ConcurrentHashMap<String, Patient> _patients** where the key is the bracelet MAC address. The Tent class provides interface for checking, addition and update of Treatments that each Patient received. The most important function is **updatePatientInfoFromBT** , which receives the main BTservice data structure – the mapping of mac-addresses to list of messages, which are actually treatments.

Each **Patient** contains **Bracelet,** which contains **Treatment**.



Bracelet: 00:14:03:05:26:9D
ID: 7654321
Treatment received: Tourniquet at 20:30 of type A
Treatment received: Acamol at 20:30 of type B
Treatment received: Morphine at 20:31 of type C
36.3 °C 98/48

In the application each patient is uniquely identified by its bracelet's mac-address. There is also an option to scan an NFC tag with id and to add a personal id to a patient, but this information is optional and don't have any influence on logic.

Every message that is received by **updatePatientInfoFromBT** is dispatched by Tent to an appropriate Patient and then to Bracelet, where with **ArduinoParsingUtils** it's being parsed and specific **Treatment** is created. A message could be also a command to update or remove existing treatment. (see Arduino messages specification in **ArduinoParsingUtils**).

Each **Treatment (a class)** is uniquely identified by a pair
<minute of treatment>|<the number of treatments in the same minute>
when semantically each treatment is identified by a uid that is written on NFC tag. This serial number is sent to web database which tells what is the treatment name and type.

Tent Activity

TentActivity displays all patients list with the preview of treatments that they've received.

It sets up a **BTservice**, an **NFCadapter** and two threads **UpdateData** and **CheckEvacuation**. It also instantiates a **Tent.**

BTservice is started at OnCreate such that scanning for Bluetooth devices begins at the very beginning of the activity.

UpdateData is the thread that synchronizes between all messages that were received by BTservice and actual logic data that consists of Patients and Treatments in application, which are in their turn used by UI, that is why the state of these structures needs to constantly updated.

CheckEvacuation periodically updates the state of evacuation status (whether the evacuation request was sent or not) of all patients and synchronizes it between the record on physical bracelet and the record on the web.

NFCadapter is used in this activity for two purposes:

- scanning an NFC tag with bracelet mac. The action opens another activity – PatientActivity, i.e. the same as click on a specific row. It mainly serves for locating a bracelet of interest on the Android device screen.
- scanning an NFC tag with patient's personal number, the action will add an ID: <number> below the bracelet mac.

Logger

Logger is a class for debug purposes, a public static instance could be initialized which will create a log file named <timestamp>101app.log under Documents directory in the device's internal storage and then a **writeToLog(String message)** could be called to register a message from anywhere in the code.

Next Steps and Enhancements

One technical challenge that needs enhancement both from Arduino and Android side is to transfer the data on treatments through NFC. The main difficulties here are combining between NFC scanner read and write modes and NFC tag simulation by both Android and Arduino NFC devices.

Note also that Bracelet uses ArduinoParsingUtils directly, further development may consider using another bracelet with different protocol and message format such that Bracelet would better use an abstract class with parsing utilities.

Web

General Description

As mentioned, all of the data is being continuously synchronized with remote servers. This is done mainly by a group of extended AsyncTask classes and a few other classes. The services being done by these classes are: creating a general treatments table, handling evacuation calls, login and logout actions, location service, and sending all of the patients data to the servers.

For any change being to the patients and users and updated in the database, we poke the servers using the **postToWeb()** function from the **PostToWeb** class. This function using http POST action to let the website know there are changes with some of the patients and/or users.

TreatmentsTable

Once the app is activated, it creates an instance of the TreatmentsTable, which is a class holding to translation tables: **codesToEquipmentTable** and **EquipmentNameToCodeTable**. On construction, this class connects to the server using **updateActivitiesTable**(…), an extended asynctask, and update the tables for codes and treatments. This class also holds **getters** by code and by equipment name.

Login and Logout

When a user connects to the app, his username and password are being checked at the users database in the servers. This is done by the **LoginTask**, which search for a corresponded user details on the database with the **checkUserAndPass(…)** function, and if ok, opens the TentActivity with the user's details. When the user is logging out, the **LogoutTask** is being activated to update the servers for him being not connected, and empty his details from the current session. Both classes are an extended asynctasks.

Location

Once the app entering the TentActivity page, it creates a location service using the **MyCurrentLocationListener** class, which implements the LocationListener interface. On every location chage, being defined for minimum distance of 10m, this class calls the **onLocationChange** which checks if the new location is new and is more accurate than the location we already know. This is done with the **isBetterLocation(…)** function. For a true result, the **latitude** and **longitude** fields being updated and sent to the user's details on the server with **LocationTask,** an extended asynctask, which connect the server and update the user's location.

Evacuation

When evacuation is being called from the app for a specific patient, it is activating the **CallEvacuationTask,** which connects to the server and updates is evacuation status. On the other hand, the app is constantly checking for evacuations being called from the web. The check is done using **CheckEvacutionTask,** which checks for any of our local patients if its evacuation status has been changed by the website. If so, updates his status on the app.

Sending Data to Web

Every detail changing in our local patients is being updated to the web servers. Including treatments, injury status, personal details, location etc. this is done by the **SendToMongodbTask**. This extended asynctask connects to the server and, for every local patient, updates all of his details into the database done at the **insertToDocAndUpdate**(…) function of the task.

Next Steps and Enhancements

One important challenge is to create better coordination between evacuation status being updated from the web and evacuation status being updated from the app. If the web is updating the status for true on the DB, and on the same time the app is updating for false, then one of them is clearly not updated.

User Interface

Login Activity

LoginActivity displays login screen.

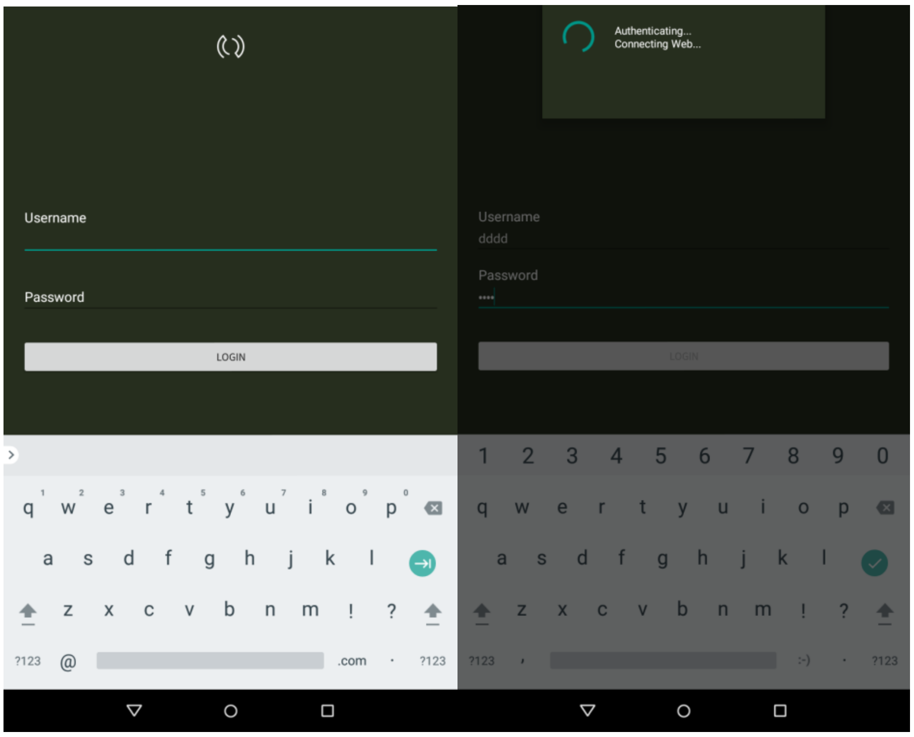
the doctor can log in according to the user name and password updated in mongoDB database.

onCeate() initializes all the views of the screen and their behavior. also, in the background, **BluetoothAdapter** is initialized and being checked if supported on the device, and **TreatmentsTable** is created, for mapping information from web into treatments.

validatePassword communicates which mongoDB server, and checks whether the password match the user name. The connection is proceed by **LoginTask**.

initiateProgressDialog

login checks the validation of the input, and opens an authentication screen while connecting to web.



Patient Information Activity

PatientInfoActivity displays treatments list chosen soldier received. Also there is option to send urgant evacuation and edit treatments.

UpdateData is the thread that synchronizes between all messages that were received by BTservice and actual logic data that consists of Treatments in application, their times, Urgent Evacuation information, and the state of the soldier.

CheckEvacuation periodically updates the state of evacuation status (whether the evacuation request was sent or not) of certain soldier.

createDialogEditTreatment creates dialog box with option to input new treatment name. The new name must be a legal one, according to the treatments in web.

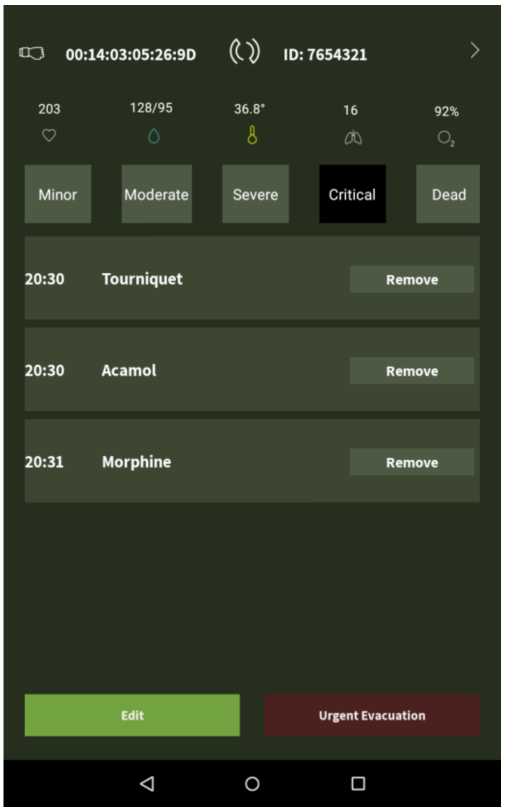
setOnClickState is the behavior of the state buttons. When clicking on soldier state button, it changes color, and the information updating in the bracelet and web through the UpdateData thread.

loadState load the initial state (if there is one) according the information in the web and bracelet.

initUrgentButton is the behavior of urgent evacuation button. When the user clicks on it, he can click on it again within one minute, and the decision will be canceled, before canceling it, creates a dialog box for asking the user permission to continue.

initSaveButton is the behavior of save/edit button. After the user finishes editing the treatments, he clicks on Save button. Then the information will be sent to web.

initListOfTreatments initiates the list of treatments according to information sent from bracelet.



00:14:03:05:26:9D ID: 7654321

203 128/95 36.8° 16 92%

Minor Moderate Severe Critical Dead

20:30	Tourniquet	Remove
20:30	Acamol	Remove
20:31	Morphine	Remove

Edit Urgent Evacuation

Next Steps and Enhancements

Extended GUI and design. Making a UI that suitable to the needs of the soldiers in the field.

Make more options to update and communicate with web and bracelet, and make the app run faster.