ARIEL UNIVERSITY / Department of Computer Science and Mathematics

MOBILAB

Abstract:

During the process of investigating helpful technologies to autonomous vehicles, we found that there is a real need in a standalone system that hold the ability to gather and process real-time information and telemetry that connected to it, such as: Camera, Microphone, Barometer, GPS Location, Temperature, etc…

The ability to process this information and send it to the ground-station in real-time, parallelly to send this information to a connected cloud services and of course store this information and data locally on the device storage, grant the autonomous vehicle technology and specifically UAV's new strong and important abilities.

Such technology can get the market to developing new autonomous vehicles with new abilities that never seen up for today, also improvement of existing systems on existing autonomous vehicles that are not carrying mobile lab systems such as describe above.

The MobiLAB application designed to include two separated components, the former will be the system Application for an Android devices which will give the user controlling abilities on recording the sensors and change the settings.
And the latter will be an Android background Service which will run on the background of the mobile device and will be the authority to record data from the sensors, gather important information, analyze it, send the data to the ground and cloud control stations.

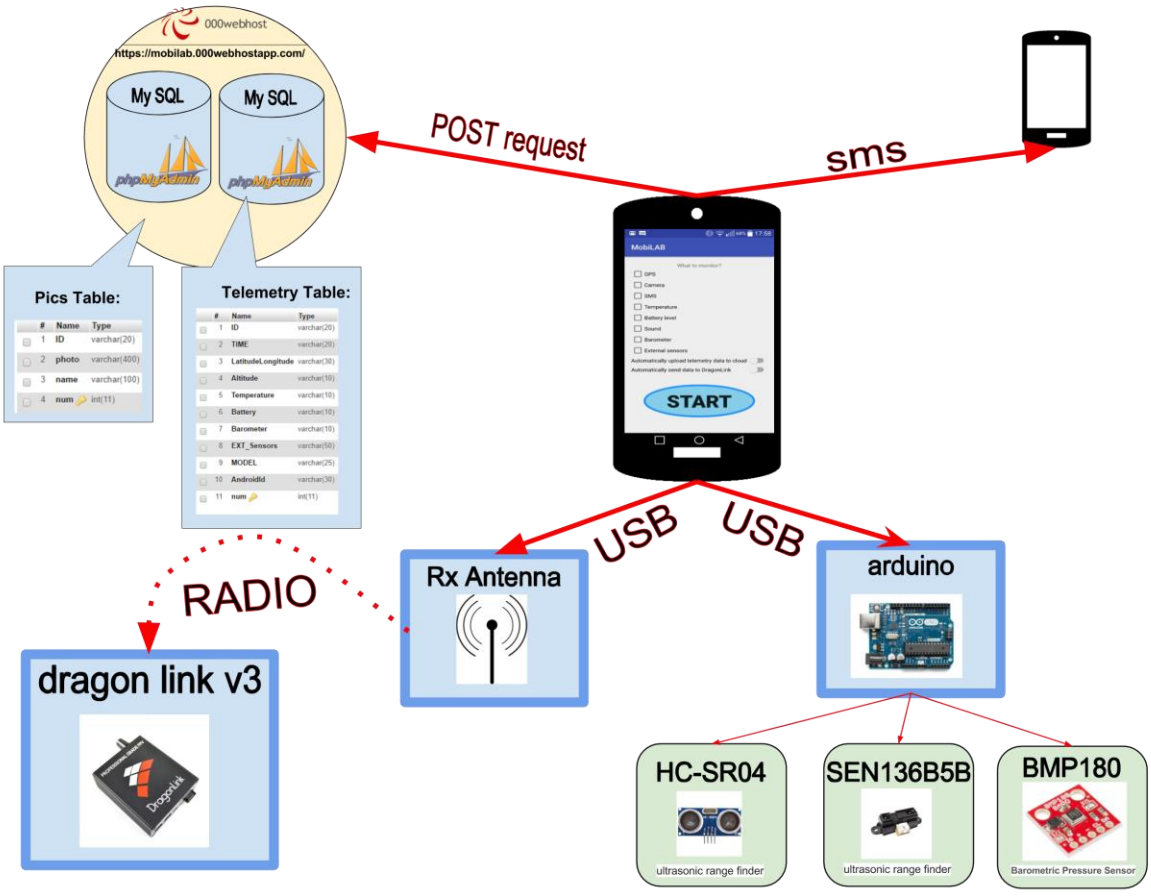
The Android device planned to be as small as possible and the application won't demand from it high computing performance of a new Android devices (Android SDK 4.4.3 version or newer), to keep things and weight as simple and light as possible.

As we said the device will communicate with the user on several channels of communication such as:

- **SMS messages** which will contain critical information about the system state such as GPS location, battery level and temperature, also they will be encoded in order to save bandwidth.
- **4/3G connectivity** to the cloud that was pre-configured in order to real-time backup all the gathered information.
- **UHF Transceiver** (DragonLink) that send and receive all the data and telemetric information from the device into the ground station.
- **Local Storage** save all of the information which will be encrypted to avoid unwanted elements to access the data and also save storage space on the disk.

Mobilab architecture

BackEnd



Define the need:

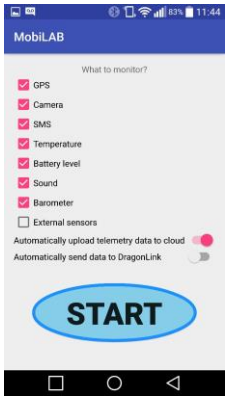
develop android system that makes the android device to “mobile lab”.
The system will be useful to scientific experience or just for people who wants to get information from their remote device.

Use cases:

- The app will use all the relevant sensors that exist on the device (GPS, Temperature, Atmospheric Pressure, Sound, etc…). ability to connect external sensors (via the auxiliary or the mini USB OTG).
- Communicating with the data to database.
- Sending SMS’s with the relevant data to the ground station.
- Every user will have separately web Dashboard database.
- The app could get orders from other devices (with SMS for example).
- The application is guaranteed to be efficient and battery friendly and totally secured.

Sign in & Authentication:

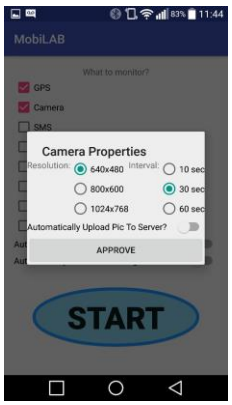
Right with the launch of the application the user will prompt to an authentication screen to fill in google username and password account.



Main Screen View:

After the user has logged into the application the main menu screen will appear and will have the following check boxes options:

- GPS
- Camera
- SMS
- Temperature
- Battery level
- Sound
- Barometer Pressure
- External sensors
- Upload data to cloud
- Using the UHF (DragonLink) ability



Launching the service:

After pressing the “START” button a service with several threads will launch and start monitoring the environment until the back key will pressed by the user.

Remote SMS control - TBA:

Controlling the application will be available remotely from another device by using SMS messages using the preconfigured template and commands. The string that will be received on the ‘lab’, will be phrased and will execute the wanted commands remotely.

SMS critical data monitoring:

Critical data about the system such as GPS location and battery level will be sent to the preconfigured number that was entered by the user.

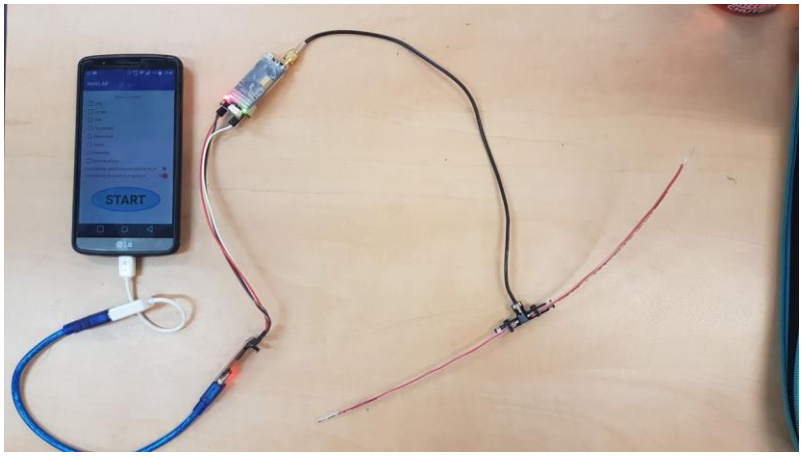
Backend access to Database:

The app will be synced with the preconfigured private cloud and will upload the recorded information directly to it, whenever the signal affords it.

Frontend access to Database:

Every user will have a private web real-time updated dashboard that will show a graphic user interface with all of the data that received from the MobiLAB application.

UHF communication (DragonLink):



Application design:

Login Activity	Main Menu Activity	Main Activity
Sign in screen	main menu screen	application working screen
username	list of sensors:	gps location parameters
password	gps	camera preview box
connect button	camera	using the CameraPreview Object
create new account	sms	using the USBService Object
	temperature	relevant threads from main menu invoked
	battery level	
	sound	
	barometer	
	external sensors	
	launch service button	

Experiments:

19.1.16 Haifa bay balloon blow –

In this experiment, we used an Israeli SIM card, and communicate through 3G communication.
We managed to receive photos from maximum altitude of ~4700.

<sms protocol="0" address="+972542594032" date="
1484819442679" type="
20170119_115030|p=32.68702
35.11471|al=4618|bt64.0|tm=25.3.jpg" toa="null"
sc_toa="null" service_center="+97254120032" read="
1" status="-1" locked="0" date_sent="1484819443000
" readable_date="Jan 19, 2017 11:50:42 AM"

SMS from 4618 meter !

YouTube Video

6.2.17 Mizpe Ramon bay balloon blow –

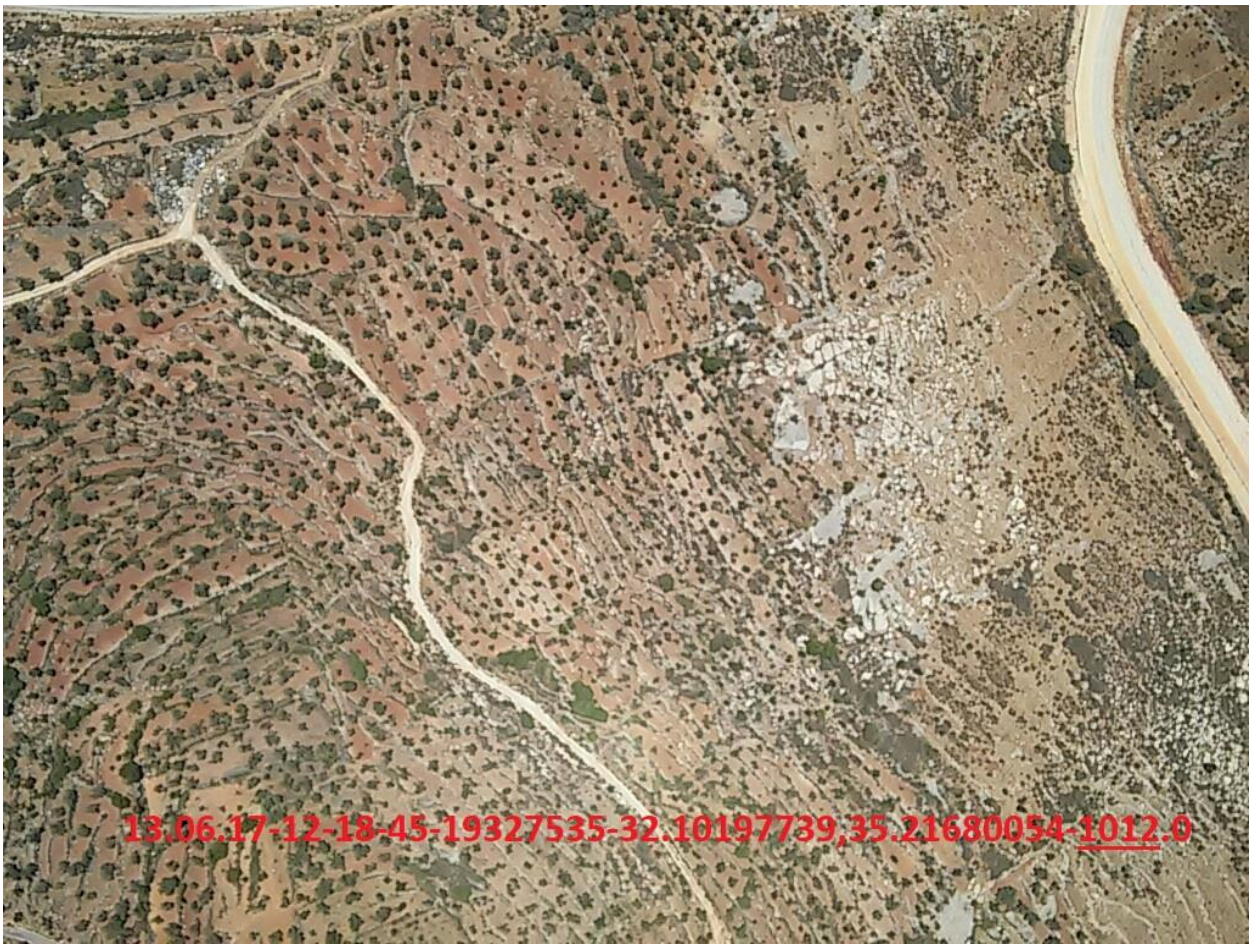
6.6.17 Ariel bay balloon blow –

In this experience we failed, we used Jordan SIM card, we had problem, all the pictures were white and all the telemetry was the same…

13.6.17 Ariel university balloon blow – we used with long WiFI signal to process the data, we got full telemetry until 1658 meter (1 kilometer rang):

ID	TIME	LatitudeLongitude	Altitude	Temperature	Battery	Barometer	EXT_Sensors	MODEL	AndroidId	num
101763630	13.06.17--12:21:53	32.09726646,35.22614143	1506.0	52.6	93.0	-1.0	0.0	LGE LG-D722	355403064309245	126544
40170488	13.06.17--12:22:03	32.09634229,35.22731127	1567.0	52.6	93.0	-1.0	0.0	LGE LG-D722	355403064309245	126545
58873232	13.06.17--12:22:13	32.09634229,35.22731127	1567.0	52.6	93.0	-1.0	0.0	LGE LG-D722	355403064309245	126546
34148608	13.06.17--12:22:23	32.09526765,35.22843129	1629.0	52.6	93.0	-1.0	0.0	LGE LG-D722	355403064309245	126547
43092854	13.06.17--12:22:33	32.09482621,35.22899521	1658.0	52.6	93.0	-1.0	0.0	LGE LG-D722	355403064309245	126548
43092854	13.06.17--12:22:33	32.09482621,35.22899521	1658.0	52.6	93.0	-1.0	0.0	LGE LG-D722	355403064309245	126549


13.06.17-12-18-14-82060841-32.10221023,35.2159649-942.0


13.06.17-12-18-45-19327535-32.10197739,35.21680054-1012.0


13.06.17-12-19-15-65933919-32.10145212,35.21763985-1096.0

5.7.17 Ariel university balloon blow – we used Israeli Sim card + added**:**

05.07.17-12_24_15_24559436_32.10427925,35.21708857_**2188.0**.jpg

05.07.17-12_27_05_91882699_32.11526637,35.22068283_**2992**.0.jpg

05.07.17-12_27_55_34403844_32.12085532,35.21962723_**3359**.0.jpg

MobiLAB logger – code examples:

In our project we used various of programming languages:

Client side:

- MobiLAB Android Application – written in JAVA
- MobiLAB dashboard web page – html, css and javascript.

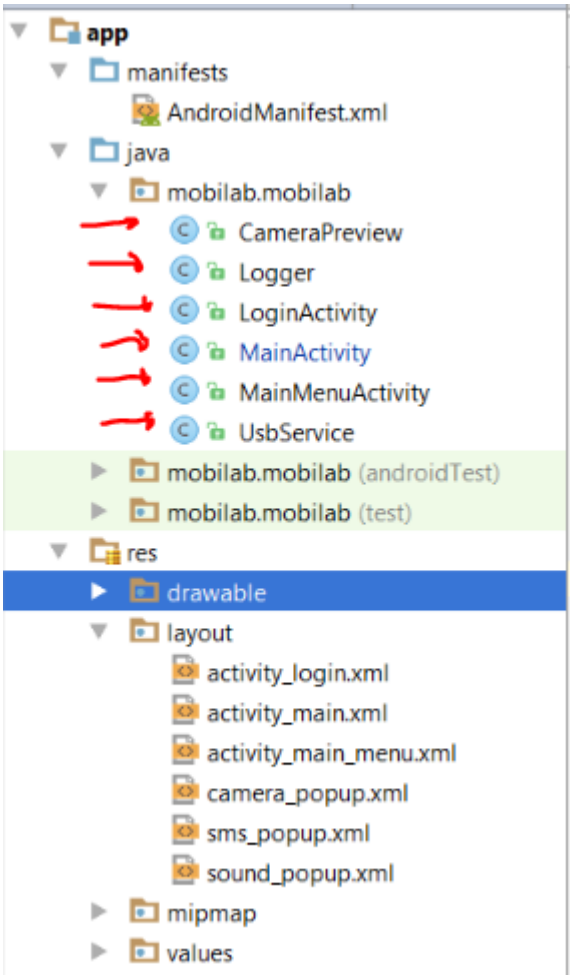
Server side:

- PHP – read and write to the D.B (Post & Get HTTP requests)
- SQL – managing D.B for telemetry and pictures, queries for specific data (By users, pictures etc…)

We also created Linux machine to store the whole project

MobiLAB Android Application:

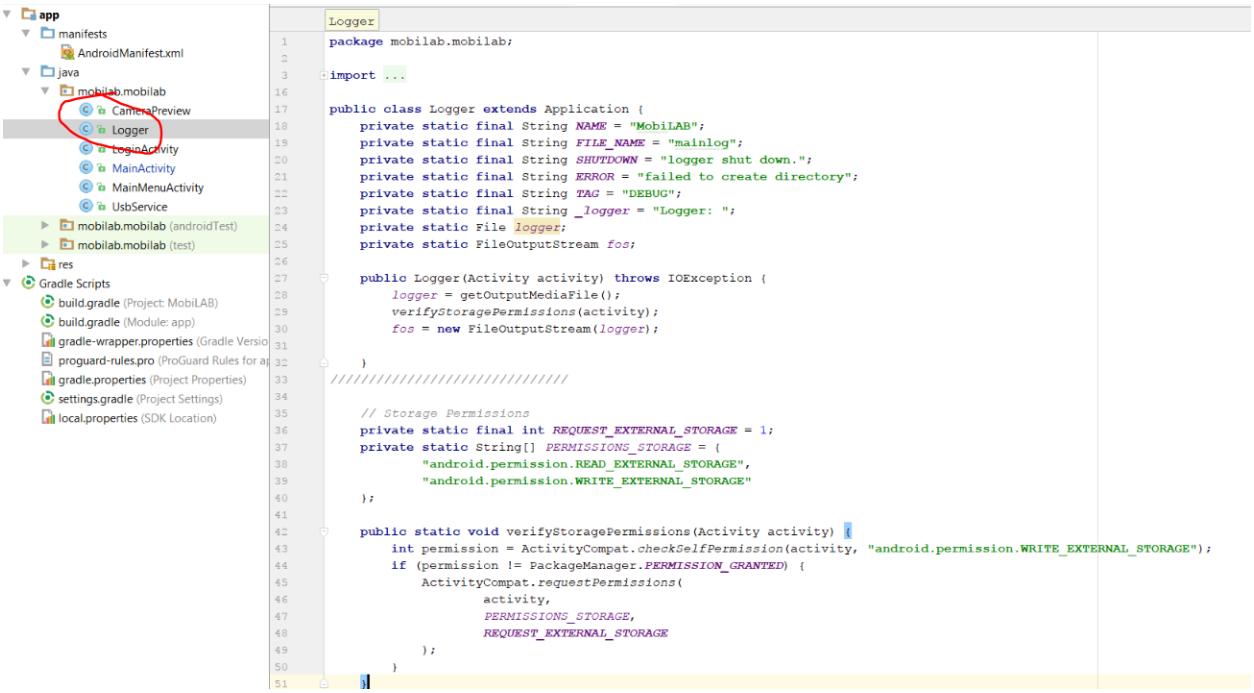
Here you can see the MobiLAB application classes:



Android manifest file:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="mobilab.mobilab">

    <!-- To auto-complete the email text field in the login form with the user's emails -->
    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
    <uses-permission android:name="android.permission.READ_PROFILE" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.SEND_SMS" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-feature android:name="android.hardware.usb.host" android:required="true"/>


    <uses-feature android:name="android.hardware.camera" />
    <uses-permission android:name="android.permission.CAMERA" />
```

MobiLAB use **Logger** class to create a text file with all the telemetry data and the necessary actions performed on the mobile:



After the user choose what to monitor, MobiLAB use different thread for each sensor in order to sampling information from the sensors.

Cloud upload Thread:

```
////////////////////////////////////////////////////////////////////////////////CloudUpload///////
//////////////////////////////////////////////////////
com.android.volley.RequestQueue requestQueue;
private static String AndroidId;
String insertUrl = "http://evron.pro/mobilab/telemetry/insertData.php";
String MODEL = Build.MANUFACTURER + " " + Build.MODEL;

Handler uploadHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        // TODO: change 0.0 values to actual values
        dataId = createID();
        currentTime = new SimpleDateFormat("dd.MM.yy--HH:mm:ss").format(new Date());
        sendToServer(dataId, currentTime, latitude, longitude, altitude,
current_temperature, current_battery_level, barometerData, 0.0, MODEL, AndroidId);
    }
};
Runnable updateCloudRunnable = new Runnable() {
    @Override
    public void run() {
        {
            synchronized (this) {
                while (CloudSwitchData) {
                    try {
                        //wait(updateCloudInterval * 1000);
                        wait(10000); //10 sec
                        uploadHandler.sendEmptyMessage(0);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                        Logger.append(e.getMessage());
                    }
                }

            }
        }
    }
};

public void sendToServer(final String ID, final String TIME, final double latitude, final
double longitude, final double altitude, final float Temperature, final float Battery,
final double Barometer, final double EXT_Sensors, final String MODEL, final String
AndroidId) {
    StringRequest request = new StringRequest(Request.Method.POST, insertUrl,
            new Response.Listener<String>() {
                @Override
                public void onResponse(String response) {
                    Logger.append("SERVER: " + response.toString());
                }
            }, new Response.ErrorListener() {

        @Override
        public void onErrorResponse(VolleyError error) {
            Logger.append("SERVER: " + error.getCause());
        }
    }) {
        @Override
        protected Map<String, String> getParams() throws AuthFailureError {
```

```
            Map<String, String> parameters = new HashMap<String, String>();
            parameters.put("ID", dataId);
            parameters.put("TIME", TIME);
            parameters.put("LatitudeLongitude", String.valueOf(latitude) + "," +
String.valueOf(longitude));
            parameters.put("Altitude", String.valueOf(altitude));
            parameters.put("Temperature", String.valueOf(Temperature));
            parameters.put("Battery", String.valueOf(Battery));
            parameters.put("Barometer", String.valueOf(Barometer));
            parameters.put("EXT_Sensors", String.valueOf(EXT_Sensors));
            parameters.put("MODEL", MODEL);
            parameters.put("AndroidId", AndroidId);
            return parameters;
        }
    };
    Logger.append("cloud updated -> ID: " + ID + ";LatitudeLongitude: " + latitude + "," +
longitude + ";Altitude: " + altitude + ";Temperature: " + Temperature + ";Battery: " +
Battery + ";Barometer: " + Barometer + ";EXT_Sensors: " + EXT_Sensors);
    requestQueue.add(request);
    Toast.makeText(getApplicationContext(), "Location Sent !", Toast.LENGTH_SHORT).show();

}
```

Automate Take Pic and upload to Server Thread:

```
////////////////////////////////////////////////////////////////////////////TakePicThread/////
//////////////////////////////////////////////////////////

    android.os.Handler handler = new android.os.Handler() {
        @Override
        public void handleMessage(Message msg) {
            try {
                mCamera.startPreview(); //important! it allow to take multi pics!
                mCamera.takePicture(null, null, mPicture);



//                mCamera = getCameraInstance();
//                mCameraPreview = new CameraPreview(getApplicationContext(), mCamera);
//                final FrameLayout preview = (FrameLayout)
findViewById(R.id.cameraPreview);
//                preview.addView(mCameraPreview);


            } catch (Exception e) {
                e.printStackTrace();
                Logger.append(e.getMessage());

            }
        }
    };


    Runnable takePicRunnable = new Runnable() {
        @Override
        public void run() {
            {
                synchronized (this) {
                    while (runPic) {
                        try {
                            wait(PICTimeOut);
                            handler.sendEmptyMessage(0);
                        } catch (InterruptedException e) {
                            e.printStackTrace();
```

```
                            Logger.append(e.getMessage());

                        }
                    }

                }
            }
        };

        /**
         * Helper method to access the camera returns null if it cannot get the
         * camera or does not exist
         *
         * @return
         */
        private android.hardware.Camera getCameraInstance() {
            android.hardware.Camera camera = null;
            try {
                camera = android.hardware.Camera.open();
            } catch (Exception e) {
                Logger.append(e.getMessage());
                Logger.append("cannot access camera or does not exist");
            }
            return camera;
        }

        Camera.PictureCallback mPicture = new Camera.PictureCallback() {
            @Override
            public void onPictureTaken(byte[] data, Camera camera) {
                File pictureFile = getOutputMediaFile();
                if (pictureFile == null) {
                    return;
                }
                try {
                    FileOutputStream fos = new FileOutputStream(pictureFile);
                    fos.write(data);
                    Logger.append("picture taken: " + pictureFile.getName());

////////////////////////////////////////////////////////////////////////////////
                    //This is very important!!!!!
                    //It return the view after taking photo!!!!!
                    mCamera.startPreview();

////////////////////////////////////////////////////////////////////////////////
                    fos.close();
                    if (uploadCameraPic) {
                        //Upload to server:
                        UpdateNewBitMap(pictureFile.getPath());
                        uploadImage();
                    }


                } catch (FileNotFoundException e) {
//                    Logger.append("can't create picture file" + e.getStackTrace());
                    Logger.append(e.getMessage());

                } catch (IOException e) {
//                    Logger.append("taking picture failed: " + e.getStackTrace());
                    Logger.append(e.getMessage());

                }
            }
        };

        private File getOutputMediaFile() {
            File mediaStorageDir = new
```

```
File(Environment.getExternalStoragePublicDirectory("MobiLAB"), "Pictures");
        if (!mediaStorageDir.exists()) {
            if (!mediaStorageDir.mkdirs()) {
                Logger.append("failed to create directory for saving pictures!");
                return null;
            }
        }
        // Create a media file name
        File mediaFile;
        mediaFile = new File(mediaStorageDir.getPath() + File.separator + getStringData() +
".jpg");
        Toast.makeText(getApplicationContext(), "new Pic created!, " + mediaFile.getName()
+ "Location: MobiLAB/Pictures", Toast.LENGTH_SHORT).show();
        Logger.append("New Pic: " + getStringData() + ".jpg");
        return mediaFile;
    }


    public void UpdateNewBitMap(String path) {
        try {
            picPath = path;
            File f = new File(picPath);
            Uri imageUri = Uri.fromFile(f);
            bitmap = MediaStore.Images.Media.getBitmap(getContentResolver(), imageUri);
        } catch (IOException e) {
            e.printStackTrace();
            Logger.append(e.getMessage());
        }
    }

    private void uploadImage() {
        //Showing the progress dialog
        final ProgressDialog loading = ProgressDialog.show(this, "Uploading...", "Please
wait...", false, false);
        StringRequest stringRequest = new StringRequest(Request.Method.POST, UPLOAD_URL,
                new Response.Listener<String>() {
                    @Override
                    public void onResponse(String s) {
                        //Disimissing the progress dialog
                        loading.dismiss();
                        //Showing toast message of the response
                        Toast.makeText(MainActivity.this, s, Toast.LENGTH_SHORT).show();
                    }
                },
                new Response.ErrorListener() {
                    @Override
                    public void onErrorResponse(VolleyError volleyError) {
                        //Dismissing the progress dialog
                        loading.dismiss();

                        //Showing toast
                        Toast.makeText(MainActivity.this, "Cant upload pic!",
Toast.LENGTH_SHORT).show();
                    }
                }) {
            @Override
            protected Map<String, String> getParams() throws AuthFailureError {
                //Converting Bitmap to String
                String image = getStringImage(bitmap);
                //Getting Image Name

                String name = getNewPicName().trim();

                //Creating parameters
                Map<String, String> params = new Hashtable<String, String>();

                //Adding parameters
```

```
                params.put("ID", dataId);
                params.put(KEY_IMAGE, image);
                params.put(KEY_NAME, name);

                //returning parameters
                return params;
            }
        };


        //Creating a Request Queue
        RequestQueue requestQueue = Volley.newRequestQueue(this);

        //Adding request to the queue
        requestQueue.add(stringRequest);

    }

    public String getStringImage(Bitmap bmp) {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        bmp.compress(Bitmap.CompressFormat.JPEG, compressQuality, baos);
        byte[] imageBytes = baos.toByteArray();
        String encodedImage = Base64.encodeToString(imageBytes, Base64.DEFAULT);
        return encodedImage;
    }

    public String getNewPicName() {
        currentTime = new SimpleDateFormat("dd.MM.yy-HH:mm:ss").format(new Date());
        if (CloudSwitchData == false) {
            dataId = createID();
        }
        String picName = currentTime + "|" + dataId + "|" + latitude + "," + longitude +
"|" + altitude + ".jpg";
        Logger.append("picName= " + picName);
        return picName;
    }
```

Automate Take SMS Thread:

```
////////////////////////////////////////////////////////////////////////////TakeSMSThread/////
/////////////////////////////////////////////////////////////

Handler handlerSMS = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        sendSms();
    }
};


Runnable runnableSMS = new Runnable() {
    @Override
    public void run() {
        //while(true)
        {
            synchronized (this) {
                while (sendSMS) {
                    try {
                        wait(SMSTimeOut);
                        handlerSMS.sendEmptyMessage(0);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                        Logger.append(e.getMessage());
```

```
                    }
                }

            }
        }
    }
};
public void sendSms() {
    SmsManager smsManager = SmsManager.getDefault();
    String SmsData = getStringData();
    smsManager.sendTextMessage(destinationNumber, null, SmsData, null, null);


    Logger.append("SMS sent to: " + destinationNumber + " Data sent = " + SmsData);
    Toast.makeText(getApplicationContext(), "SMS set to: " + destinationNumber,
Toast.LENGTH_SHORT).show();
}
```

Batterty and Tempeture Thread:

```
////////////////////////////////////////////////////////////////////////////////BATTERY and
TEMPERATURE Thread//////////////////////////////

private void batteryAndTemperatureSample() {
    Intent batteryIntent = registerReceiver(null, new
IntentFilter(Intent.ACTION_BATTERY_CHANGED));
    int level = batteryIntent.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
    int scale = batteryIntent.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
    current_temperature = ((float)
batteryIntent.getIntExtra(BatteryManager.EXTRA_TEMPERATURE, 0) / 10);
    // Error checking that probably isn't needed but I added just in case.
    if (level == -1 || scale == -1) {
        current_battery_level = 50.0f;
    }
    current_battery_level = ((float) level / (float) scale) * 100.0f;
    Logger.append("current battery level -> " + current_battery_level);
    Logger.append("current temperature -> " + current_temperature);
}

Runnable runnableBT = new Runnable() {
    @Override
    public void run() {
        {
            synchronized (this) {
                while (_battery || _temperature) {
                    try {
                        wait(BAT_TEMP_INTERVAL * 1000);
                        batteryAndTemperatureSample();

                    } catch (InterruptedException e) {
                        e.printStackTrace();
                        Logger.append(e.getMessage());
                    }
                }

            }
        }
    }
};
```

Barometer Thread:

```
////////////////////////////////////////////////////////////////////////////////Barometric
Thread/////////////////////////////////////////
Handler HandlerBarometric = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        SensorManager sensorManager = (SensorManager)
getSystemService(Service.SENSOR_SERVICE);
        final Sensor pS = sensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE);
        if (pS == null) {
            barometerData = -1;
        } else {
            barometerData = pS.getPower();
        }
        Logger.append("Barometer Pressure:= " + barometerData);
    }
};


Runnable runnableBarometric = new Runnable() {
    @Override
    public void run() {
        {
            synchronized (this) {
                while (barometerOn) {
                    try {
                        wait(barometerTimeOut * 1000);
                        HandlerBarometric.sendEmptyMessage(0);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                        Logger.append(e.getMessage());

                    }
                }

            }
        }
    }
};
```

Dragon Link Thread:

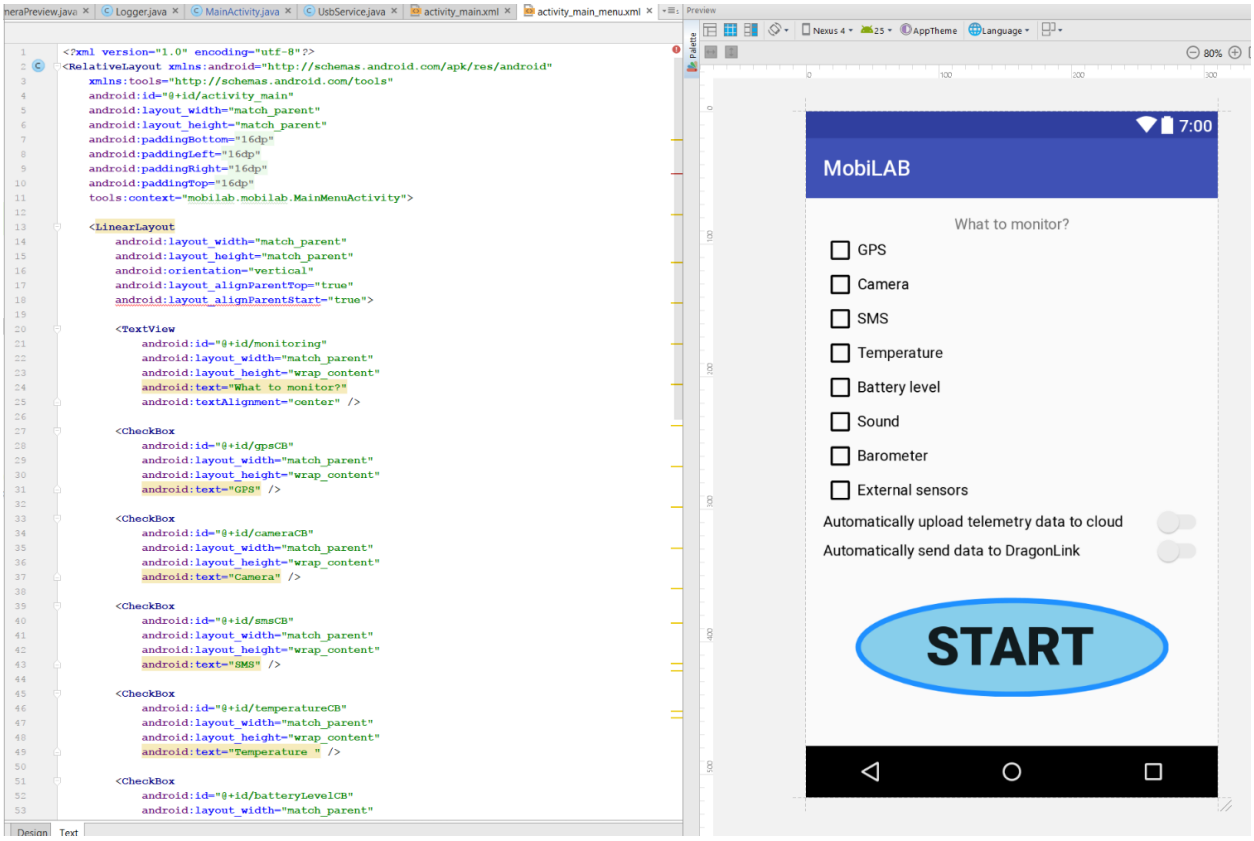
```
////////////////////////////////////////////////////////Dragon Link Thread
//////////////////////////////////////
android.os.Handler DragonHandler = new android.os.Handler() {
    @Override
    public void handleMessage(Message msg) {
        try {
            if (UsbService.class != null) { // if UsbService was correctly binded, Send
data
                UsbService.write((getStringDataShort() + "---").getBytes());
            }
        } catch (Exception e) {
            e.printStackTrace();
            Logger.append(e.getMessage());

        }
    }
};
```

```
Runnable sendToDragonMsg = new Runnable() {
    @Override
    public void run() {
        {
            synchronized (this) {
                while (true) {
                    try {
                        wait(2000);
                        DragonHandler.sendEmptyMessage(0);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                        Logger.append(e.getMessage());

                    }
                }

            }
        }
    }
};
```

UsbService Class:

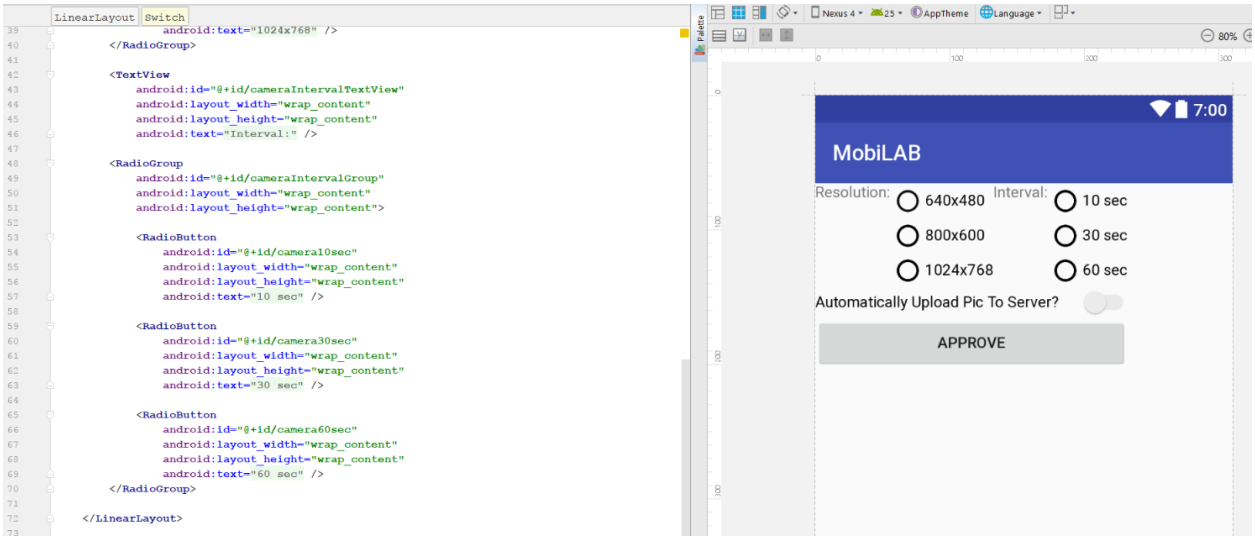
```
import ...

public class UsbService extends Service {

    public static final String ACTION_USB_READY = "com.felhr.connectivityservices.USB_READY";
    public static final String ACTION_USB_ATTACHED = "android.hardware.usb.action.USB_DEVICE_ATTACHED";
    public static final String ACTION_USB_DETACHED = "android.hardware.usb.action.USB_DEVICE_DETACHED";
    public static final String ACTION_USB_NOT_SUPPORTED = "com.felhr.usbservice.USB_NOT_SUPPORTED";
    public static final String ACTION_NO_USB = "com.felhr.usbservice.NO_USB";
    public static final String ACTION_USB_PERMISSION_GRANTED = "com.felhr.usbservice.USB_PERMISSION_GRANTED";
    public static final String ACTION_USB_PERMISSION_NOT_GRANTED = "com.felhr.usbservice.USB_PERMISSION_NOT_GRANTED";
    public static final String ACTION_USB_DISCONNECTED = "com.felhr.usbservice.USB_DISCONNECTED";
    public static final String ACTION_CDC_DRIVER_NOT_WORKING = "com.felhr.connectivityservices.ACTION_CDC_DRIVER_NOT_WORKING";
    public static final String ACTION_USB_DEVICE_NOT_WORKING = "com.felhr.connectivityservices.ACTION_USB_DEVICE_NOT_WORKING";
    public static final int MESSAGE_FROM_SERIAL_PORT = 0;
    private static final String ACTION_USB_PERMISSION = "com.android.example.USB_PERMISSION";
    //private static final int BAUD_RATE = 9600; // BaudRate. Change this value if you need
    private static final int BAUD_RATE = 4800; // BaudRate. Change this value if you need
    public static boolean SERVICE_CONNECTED = false;

    private IBinder binder = new UsbBinder();

    private Context context;
    private Handler mHandler;
    private UsbManager usbManager;
    private UsbDevice device;
    private UsbDeviceConnection connection;
    private static UsbSerialDevice serialPort;

    private boolean serialPortConnected;
    /*
     *  Data received from serial port will be received here. Just populate onReceivedData with your code
     *  In this particular example. byte stream is converted to String and send to UI thread to
     *  be treated there.
     */
    private UsbSerialInterface.UsbReadCallback mCallback = (arg0) -> {
            try {
                String data = new String(arg0, "UTF-8");
                if (mHandler != null)
                    mHandler.obtainMessage(MESSAGE_FROM_SERIAL_PORT, data).sendToTarget();
            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
                Logger.append(e.getMessage());

            }
    };
    /*
     * Different notifications from OS will be received here (USB attached, detached, permission responses...)
     * About BroadcastReceiver: http://developer.android.com/reference/android/content/BroadcastReceiver.html
     */
    private final BroadcastReceiver usbReceiver = (arg0, arg1) -> {
            if (arg1.getAction().equals(ACTION_USB_PERMISSION)) {
                boolean granted = arg1.getExtras().getBoolean(UsbManager.EXTRA_PERMISSION_GRANTED);
```

Example of our XML files:



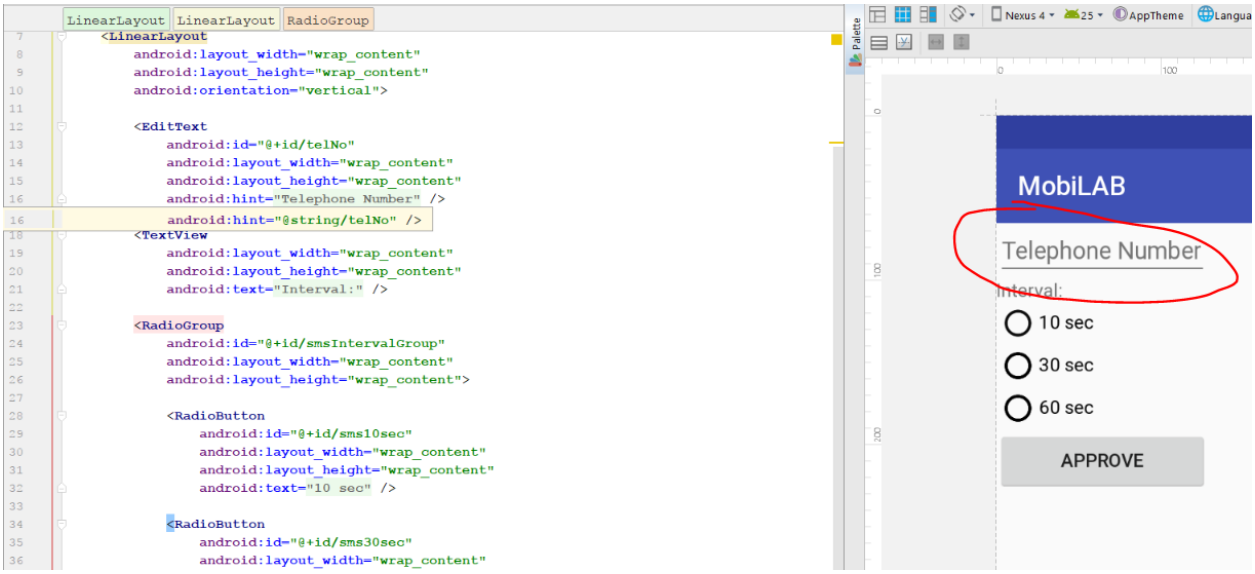
```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="mobilab.mobilab.MainMenuActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:layout_alignParentTop="true"
        android:layout_alignParentStart="true">

        <TextView
            android:id="@+id/monitoring"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="What to monitor?"
            android:textAlignment="center" />

        <CheckBox
            android:id="@+id/gpsCB"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="GPS" />

        <CheckBox
            android:id="@+id/cameraCB"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Camera" />

        <CheckBox
            android:id="@+id/smsCB"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="SMS" />

        <CheckBox
            android:id="@+id/temperatureCB"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Temperature " />

        <CheckBox
            android:id="@+id/batteryLevelCB"
            android:layout_width="match_parent"
```


Linear Layout (Camera):



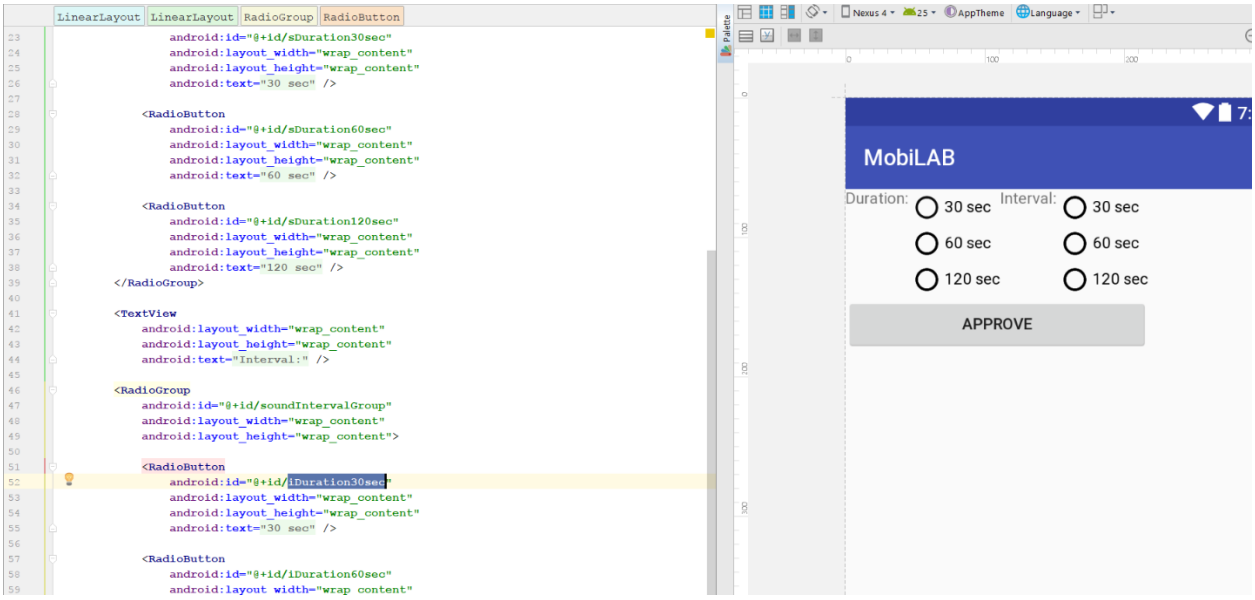
```xml
            android:text="1024x768" />
        </RadioGroup>

        <TextView
            android:id="@+id/cameraIntervalTextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Interval:" />

        <RadioGroup
            android:id="@+id/cameraIntervalGroup"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">

            <RadioButton
                android:id="@+id/camera10sec"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="10 sec" />

            <RadioButton
                android:id="@+id/camera30sec"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="30 sec" />

            <RadioButton
                android:id="@+id/camera60sec"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="60 sec" />
        </RadioGroup>

    </LinearLayout>
```

Linear Layout (SMS):



```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical">

    <EditText
        android:id="@+id/telNo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="Telephone Number" />
        android:hint="@string/telNo" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Interval:" />

    <RadioGroup
        android:id="@+id/smsIntervalGroup"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">

        <RadioButton
            android:id="@+id/sms10sec"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="10 sec" />

        <RadioButton
            android:id="@+id/sms30sec"
            android:layout_width="wrap_content"
```

Linear Layout (Sound recording):



```
            android:id="@+id/sDuration30sec"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="30 sec" />

        <RadioButton
            android:id="@+id/sDuration60sec"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="60 sec" />

        <RadioButton
            android:id="@+id/sDuration120sec"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="120 sec" />
    </RadioGroup>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Interval:" />

    <RadioGroup
        android:id="@+id/soundIntervalGroup"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">

        <RadioButton
            android:id="@+id/iDuration30sec"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="30 sec" />

        <RadioButton
            android:id="@+id/iDuration60sec"
            android:layout_width="wrap_content"
```

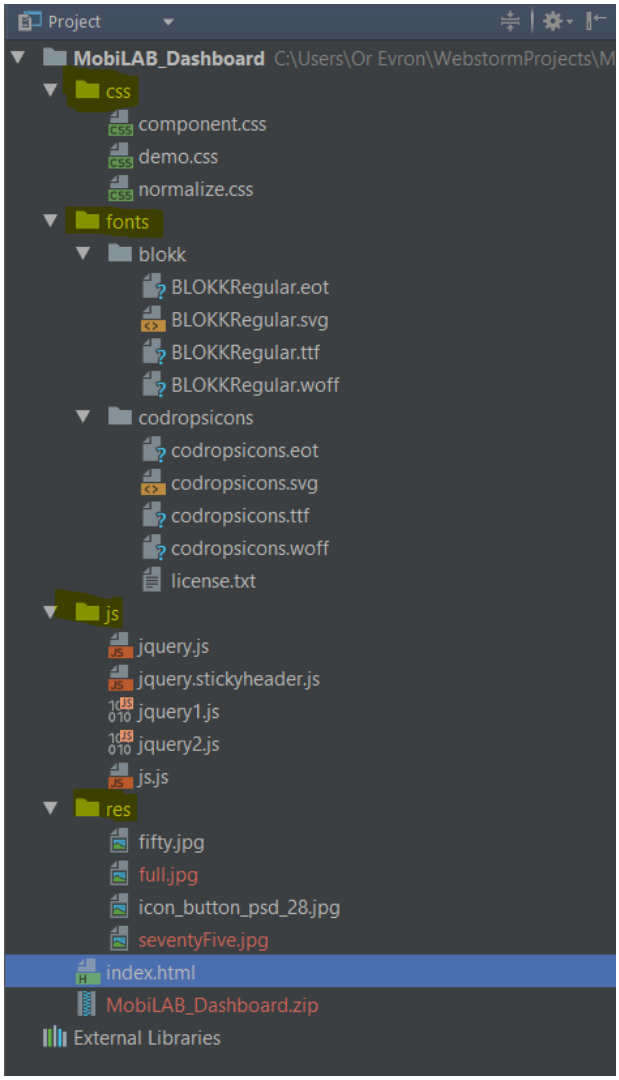
MobiLAB web page – html, css and javascripts:

We created out website dashboard with the help of a free dashboard template service called **Admin LTE,** we adopt the dashboard tools we needed in order to create out dashboard and customized it to fit our data.

This is the Dashboard project tree:

Where you can find the css's fonts and javascripts that behind the visualization of our dashboard.

Also you can see the js.js file and the index.html file which holds the customization of the template to our needs



This is part of the js.js file that communicate with the sql server via php files and receive json files as response, with those json files that represent the response query we are drawing the table on the web dashboard.

```
/**
 * Created by Or Evron on 5/30/2017.
 */
const telemetryUrl = "http://evron.pro/mobilab/queries/telemery-json.php",
    picsUrl = "http://evron.pro/mobilab/queries/telemetryPics-json.php",
    devicesUrl = "http://evron.pro/mobilab/queries/telemetry-devices.php";

function drawDB(tableName) {...}

function getJson1(tableName) {
    var $table = $(".myTable1");
    $table.html("");
    if (tableName === "Telemetry DB") {
        $.getJSON(telemetryUrl, {}, function (json) {...}
        );
    }
    else if (tableName === "Pics DB") {
        $.getJSON(picsUrl, {}, function (json) {...});
    }
    else if (tableName === "") {
        tableName = "Telemetry ID's: ";
        $.getJSON(devicesUrl, {}, function (json) {...});
    }
}

function drawTable1(json, $table, tableName) {...}

function getTableHeader1(json, $table, tableName) {
    var $header1 = $("<thead></thead>");
    var $header = $("<tr></tr>");
    $.each(json[tableName][0], function (item) {
        $header.append($("<th></th>").text(item));
    });
    $header1.append($header);
    $table.append($header1);
    $table.append("<tbody>");
    getTable1(json, $table, tableName);
}
function getTable1(json, $table, tableName) {
    json[tableName].forEach(function (item) {
        getTableData1(item, $table, tableName);
    });
    $table.append("</tbody>");
}

function getTableData1(item, $table, tableName) {...}
```

PHP – read and write for the D.B (Post & Get HTTP requests):

With PHP script files that stored on our server we are querying the sql server and insert data into it, both the mobile and the web using those scripts, one to upload files and the other to retrieve information.

This is our connection script to first connect into the sql server:

```php
<?php
header('Access-Control-Allow-Origin: *');
define('hostname', 'localhost');
define('user', '[illegible]');
define('password', '[illegible]');
define('databaseName', 'mobilabdb');
$connect = mysqli_connect(hostname,user,password,databaseName);
?>
```

This script establish connection into the sql (using the previous script) and then querying the database to get all the telemetry information:

```php
<?php
include 'connection.php';
run();

function run(){
        global $connect;
        $query = " Select * FROM telemetry; ";
        $result = mysqli_query($connect, $query);
        $number_of_rows = mysqli_num_rows($result);
        $temp_array = array();
        if($number_of_rows > 0) {
                while ($row = mysqli_fetch_assoc($result)) {
                        $temp_array[] = $row;
                }
        }
        header ('Content-Type: application/jjson');
        echo json_encode(array("Telemetry DB"=>$temp_array));
        mysqli_close($connect);
}
?>
```

The Server – Manage High availability Server alongside support the android application:

Picking the best storage services wasn't that easy, as we wanted to keep our services free of charge, we also needed to find the best **free** but yet highly available storage service, which will give us a web server including database server and php installed on to communicate between all those components.

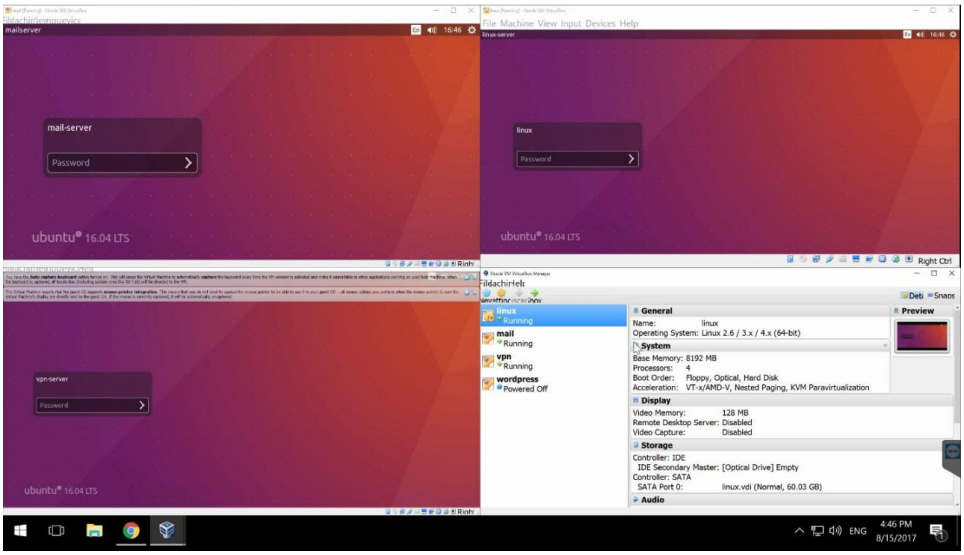
Out first choice was the site 000webhost which seem to answer all of our needs, but overall with a lot of experiments and days of testing, we found their service **NOT** highly available at all.

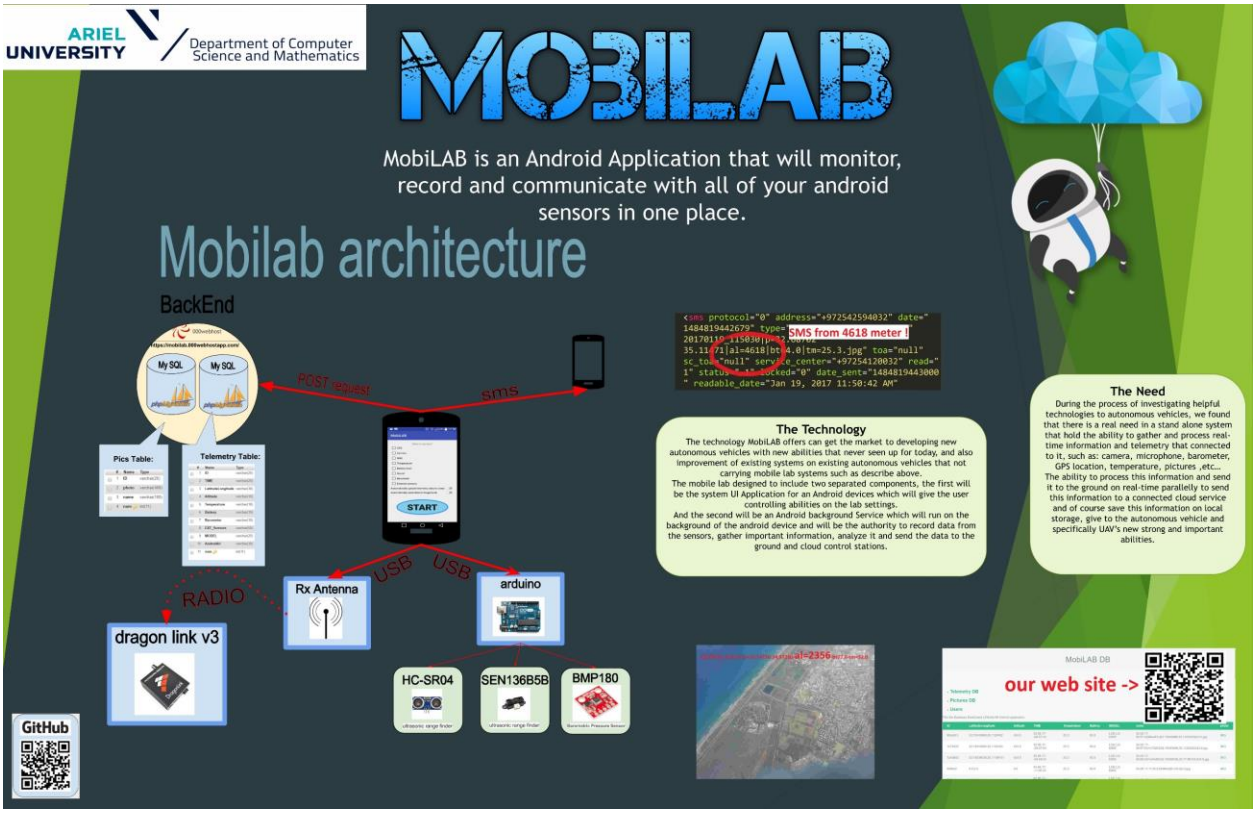
We needed another solution so we decided to take it in and serve the files on our own site.

We built a dedicated machine with a suitable hardware and configure all the configuration that needs to be done including:

- Set a static IP on our internet provider
- Buying a domain for our dashboard
- Route all traffic to its destination (VM on the dedicated machine)
- Finally built the VM's to listen to the traffic and serve our dashboard to the outside world

In order to keep our site secured as much as possible we also created a VPN machine in our site, that gives us the ability to manage users that can access our backend.




ARIEL UNIVERSITY
Department of Computer Science and Mathematics
MOBILAB
MobiLAB is an Android Application that will monitor, record and communicate with all of your android sensors in one place.
Mobilab architecture
BackEnd
My SQL
My SQL
POST request
sms
Pics Table:
Telemetry Table:
START
RADIO
USB
USB
Rx Antenna
arduino
dragon link v3
HC-SR04
SEN136B5B
BMP180
GitHub
SMS from 4618 meter !
The Technology
The technology MobiLAB offers can get the market to developing new autonomous vehicles with new abilities that never seen up for today, and also improvement of existing systems on existing autonomous vehicles that not carrying mobile lab systems such as describe above.
The mobile lab designed to include two separated components, the first will be the system UI Application for an Android devices which will give the user controlling abilities on the lab settings.
And the second will be an Android background Service which will run on the background of the android device and will be the authority to record data from the sensors, gather important information, analyze it and send the data to the ground and cloud control stations.
The Need
During the process of investigating helpful technologies to autonomous vehicles, we found that there is a real need in a stand alone system that hold the ability to gather and process real-time information and telemetry that connected to it, such as: camera, microphone, barometer, GPS location, temperature, pictures ,etc...
The ability to process this information and send it to the ground on real-time parallelly to send this information to a connected cloud service and of course save this information on local storage, give to the autonomous vehicle and specifically UAV's new strong and important abilities.
MobiLAB DB
our web site ->
Telemetry DB
Pictures DB
Users