izzati Documentation

Release 1.0.0

Gustav Hansen

Sep 03, 2017

Contents:

Source: https://github.com/kusti8/izzati

CHAPTER 1

Why?

I created this because I was recently at a hackathon where we created an app and a backend and had to send a picture to the backend and receive some text as a response. We spent close to 9 hours of our 36 hours and we still couldn't get it working. We were sleep deprived, there were so many options and we just couldn't get it to work. So I made this, which is designed to Just Work between a backend and a frontend where you can send send text or a file with a few commands and removing all the need for extra research or dealing with HTTP POST or GET etc.

Features

- Dead simple library that is intuitive to use
- Supports sending images back and forth and returns a simple file object on both sides
- Uses callbacks so no large while loop
- Deciphers all the messages into native dictionary objects
- Clear documentation so everything you need is in one place

CHAPTER 2

Quickstart - Backend

Installation

```
sudo pip install izzati
```

Usage

A simple example which prints out the data received and a list of the filenames of all the files which were received.

```
from izzati import Backend, file_return

def p(text, data):
    print(text, [x.filename for x in data])
    return {'status': 'yes'}
    # return file_return('file.txt')

b = Backend(p)
b.run() # The host and port can be set as well:
# b.run(host='0.0.0.0', port=5030) # Default is 0.0.0.0 and 5020
```

The function must have arguments for text and data, even if no data is sent.

A file can also be returned with the file_return function.

Data

The data supports many functions that are revealed:

`file.read()` Read a file `file.save(location)` Save a file to a location `file.filename` The name of the file when it was transferred

Text

The text is all returned as a dictionary, with keys and values. During transit, it is converted to JSON, but will always be exposed as a dictionary.

Background Processes

Izzati provides a nice frontend for background processes and supports running the backend in the background or a function in the background. This is built on top of multiprocessing and supports multiple CPU cores etc.

1. Make the webserver in the background.

```
from izzati import Backend

def callback(q, text, data): # <----------------------------------
    print(text, [x.filename for x in data])
    return {'status': 'OK'}

b = Backend(callback, background=True) # <------------------------
p, q = b.run() # <-------------------------------------
# Block the Python process, or else everything quits
```

Notice there's two extra return values, p and q and the callback accepts an extra argument. This is a Queue object, where you can communicate with the callback function. Use `q.put("hello")` to put data into the queue and `q.get()` to get data out of the queue. p is the process, which can be used to join (`p.join()`) or to kill etc.

See https://docs.python.org/3/library/multiprocessing.html for more details.

The callback, in this case, must have q as the first argument, with no default value.

2. Make another process in the background.

```
from izzati import Backend, background # <-------------

def callback(text, data, q):
    print(text, [x.filename for x in data])
    q.put(text['name'])
    return {'status': 'OK'}

def background_process(q, greeting): # <---------------
    while True:
        name = q.get()
        print(greeting + ", " + name)

p, q = background(background_process, args=("Hello",)) # <--------------
# Also supports default arguments with kwargs:
# p, q = background(background_process, kwargs={"greeting": "Hello"})

b = Backend(callback, args=(q,)) # <-----------------------
# Also supports default arguments with kwargs:
# b = Backend(callback, kwargs={'q': q})

b.run()
```

These methods can be used at the same time as well, but something must be blocking the Python code from finishing.

Arguments

The Backend object supports args and kwargs, as shown above. The background process does as well. In both, the args and kwargs follow the essentials, such as text, data, and q if running in the background.

CHAPTER 3

Quickstart - Java

Install

Add the following to your project build.gradle

```
dependencies {
    compile 'com.github.kusti8:izzati:1.2.0'
}
```

Usage

The Java library is very easy to use as well

```
public void sendMessage(View view) throws JSONException {
    EditText text = (EditText)findViewById(R.id.jsonTxt);
    String value = text.getText().toString(); // The JSON data is stored here
                                              // The input parameter can be a␣
↪JSONObject or a string
    Izzati i = new Izzati();                  // Initialize Izzati
    JSONObject obj = new JSONObject(value);
    i.url = "http://192.168.100.118:5020/";   // Set in Izzati, the URL
    if (file == null) {
        i.send(obj, new IzzatiJsonHandler() { // If there is no file selected, send␣
↪JSON and expect JSON back
            @Override
            public void callback(JSONObject response) {
                System.out.println(response); // Print out the response
            }
        });
    } else {
        i.send(obj, file, new IzzatiJsonHandler() { // If there is a file, simply␣
↪specify it after the JSON
```

```
            @Override
            public void callback(JSONObject response) {
                System.out.println(response);
            }
        });
    }
}
```

Return Types

If you want to receive a binary instead of a JSON, observe the following changes:

```
public void sendMessage(View view) throws JSONException {
    EditText text = (EditText)findViewById(R.id.jsonTxt);
    String value = text.getText().toString();
    Izzati i = new Izzati();
    JSONObject obj = new JSONObject(value);
    i.url = "http://192.168.100.118:5020/";
    if (file == null) {
        i.send(obj, new FileAsyncHttpResponseHandler(this) {
            @Override
            public void onFailure(int statusCode, Header[] headers, Throwable␣
↪throwable, File file) {
                throwable.printStackTrace();
            }

            @Override
            public void onSuccess(int statusCode, Header[] headers, File response) {
                Log.w("self", response.toString());
            }
        });
    } else {
        i.send(obj, file, new FileAsyncHttpResponseHandler(this) {
            @Override
            public void onFailure(int statusCode, Header[] headers, Throwable␣
↪throwable, File file) {
                throwable.printStackTrace();
            }

            @Override
            public void onSuccess(int statusCode, Header[] headers, File response) {
                Log.w("self", response.toString());
            }
        });
    }
}
```

NB! Remember Permissions

Remember to set permissions in the manifest:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

And to request permissions if you are accessing files:

```
private void checkPermission(){
    if (checkSelfPermission(Manifest.permission.READ_EXTERNAL_STORAGE)
            != PackageManager.PERMISSION_GRANTED) {

        // Should we show an explanation?
        if (shouldShowRequestPermissionRationale(
                Manifest.permission.READ_EXTERNAL_STORAGE)) {
            // Explain to the user why we need to read the contacts
        }

        requestPermissions(new String[]{Manifest.permission.READ_EXTERNAL_STORAGE},
                123);

        // MY_PERMISSIONS_REQUEST_READ_EXTERNAL_STORAGE is an
        // app-defined int constant that should be quite unique

        return;
    }
}
```

And then call that function in the onCreate override.

CHAPTER 4

Quickstart - Python

Install

The frontend is included with the backend, so a simple `sudo pip install izzati` will work

Usage

Usage is dead simple:

```
from izzati import Frontend

f = Frontend('http://localhost:5020/') # Initialize the frontend with a url
f.send(js={'test': '123'}, f=open('/tmp/test.jpg', 'rb')) # Send a dictionary and a␣
↪file
```

A dictionary doesn't need to be sent:

```
from izzati import Frontend

f = Frontend('http://localhost:5020/') # Initialize the frontend with a url
f.send(f=open('/tmp/test.jpg', 'rb')) # Send a file
```

That's it!

CHAPTER 5

Quickstart - React Native

Install

```
npm i react-native-izzati --save
RNFB_ANDROID_PERMISSIONS=true react-native link react-native-fetch-blob
```

Usage

1. Import it `import Izzati from 'react-native-izzati'`
2. Then create a new instance with the URL `let i = new Izzati("http://192.168.1.17:5020/")`

3. Then send, using `i.send(options)`. The format is described in the below table.

The input options object has three items which are all themselves objects, text, file, and response. Not all of them must be used. All are optional.

text	file	response	notes
key			The dict key
value			The dict value
	uri		A path
	base64		A base64 representation of the data, **without** proper prefix
	filename		The filename of the file
		base64	true or false. If false, then a path is outputted

An example would be `{text: {hello: 'me'}, file: {uri: resizedImageUri, filename: 'photo.jpg'}, response: {base64: false}}`

The output object has three possible outputs, text, base64, or path.

text	This is a normal JS object, when text is returned.
base64	This is the base64 representation of a file, without the prefix.
path	This is the path of the file, without the required file:// on Android.

The callback only takes one argument, and that is the out object.

A few functions exist other than send.

prefixJpg(base64)

Adds '' to base64 and returns the result.

prefixPath(path)

Adds 'file://' if on Android and returns the result.

Example

```
let i = new Izzati("http://192.168.1.17:5020/")
i.send({text: {hello: 'me'}, file: {uri: resizedImageUri, filename: 'photo.jpg'},␣
↪response: {base64: false}}).then( out => {
    this.setState(previous => {
        return {uri: i.prefixPath(out.path)}
  })
}).catch(err => {
    console.log(err)
})
...
<Image style={{flex: 1}} source={{uri: this.state.uri}} />
```

CHAPTER 6

Quickstart - ESP8266

Install

Add the following to the top of your code:

```
#include <ESP8266HTTPClient.h>
#include <ArduinoJson.h> // Only if working with JSON. Use https://bblanchon.github.
↪io/ArduinoJson/assistant/ for size.

HTTPClient http;
```

And the following function:

```
String izzati(String url, String data) {
    http.begin(url);
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");
    http.POST(data);
    String str = http.getString();
    int locStart = str.indexOf("\"");
    if (locStart==-1) return "";
    locStart += String("\"").length();
    int locFinish = str.indexOf("\"", locStart);
    if (locFinish==-1) return "";
    return str.substring(locStart, locFinish);
}

JsonObject& izzatiJson(String url, String data) {
    const size_t MAX_CONTENT_SIZE = 512;
    const size_t BUFFER_SIZE = JSON_OBJECT_SIZE(2) + JSON_ARRAY_SIZE(2) + MAX_CONTENT_
↪SIZE; // See: https://bblanchon.github.io/ArduinoJson/assistant/
    http.begin(url);
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");
    http.POST(data);
    String str = http.getString();
    DynamicJsonBuffer jsonBuffer(BUFFER_SIZE);
```

```
    JsonObject& root = jsonBuffer.parseObject(str);
    return root;
}
```

Usage

For string: `String out = izzati("http://192.168.100.113:5020/", "test=ing");` sends a message that would look like `{'test': 'ing'}`. **The output from the server must be a string.**

For JSON:

```
JsonObject& ing = izzatiJson("http://192.168.100.113:5020/", "test=ing");
String hello = ing["hello"];
```

CHAPTER 7

Quickstart - Other

Usage

In whatever programming language you use, notice the following protocol:

All data is sent using parameters, in a JSON-like way. So for example, a parameter with key “hi” and value “me”, would exit the backend as {“hi”: “me”}.

Data is sent as a parameter as well, *with key ‘file’*.

Everything is a POST request to a URL. The sent message is either data, a file, or both. The return is either data or a file.

That’s it!

CHAPTER 8

Samples

Backend

```
from izzati import Backend, file_return

def pr(form, files):
    print(form) # Print out the dictionary received
    for x in files:
        x.save('/tmp/test.jpg') # Save the file received
    return {'It': 'Worked'} # Return a dictionary to the frontend

back = Backend(pr)
back.run()
```

Frontend - Python

```
from izzati import Frontend

f = Frontend('http://localhost:5020/')
out = f.send(js={'test': '123'}, f=open('/tmp/test.jpg', 'rb')) # Send a file and a␣
↪dictionary
print(out)
```

Frontend - Java

```
public void sendMessage(View view) throws JSONException {
    EditText text = (EditText)findViewById(R.id.jsonTxt);
    String value = text.getText().toString(); // The JSON data is stored here
                                              // The input parameter can be a␣
↪JSONObject or a string
```

```
    Izzati i = new Izzati();                    // Initialize Izzati
    JSONObject obj = new JSONObject(value);
    i.url = "http://192.168.100.118:5020/";    // Set in Izzati, the URL
    if (file == null) {
        i.send(obj, new IzzatiJsonHandler() { // If there is no file selected, send␣
↪JSON and expect JSON back
            @Override
            public void callback(JSONObject response) {
                System.out.println(response); // Print out the response
            }
        });
    } else {
        i.send(obj, file, new IzzatiJsonHandler() { // If there is a file, simply␣
↪specify it after the JSON
            @Override
            public void callback(JSONObject response) {
                System.out.println(response);
            }
        });
    }
}
```

CHAPTER 9

Heroku

Install

Deploy the following:

Usage

Create a file on a publically hosted place, with a single function, called "callback". Like the backend quickstart, it takes all the same arguments and operates the same way. It **must be called "callback"**. Specify the URL in the heroku deploy, and you're all set.

Everytime a new call is made, the script will check for changes, and if there are, will run the latest code available to download.

Your changes may take a few minutes to update. For rapid debugging, it is best to just start a quick bash shell: ''heroku run bash'' and experiment there.

Own Changes

```
heroku login
heroku git:clone -a [name of deploy]
cd [name of deploy]
# Make whatever changes you need
git add .
git commit -m "Added changes"
git push heroku master
```

Be sure to change Procfile to reflect the name of your new script

CHAPTER 10

Indices and tables

- genindex
- modindex
- search