

**Touch Display: Main Unit**

**Internship**

Submitted in Fulfilment of the

Requirements for the Internship

Dept. of Computer Science

Chair of Computer Engineering

Submitted by: Abdul Rahman Idrees

Student ID: 434371

Date: 23.09.2018

Supervising tutor: Prof. Dr. W. Hardt

Dr.Ariane Heller

Abstract

Graphical User Interface (GUI) has become an important part of most applications. Some years ago, users were relying mostly on command line as the main tool of interacting with their application such as entering the inputs and reading the outputs, this has changed. In 1973 a personal computer was developed at Xerox PARC which used the concept of a Desktop and a mouse as an input tool. In 1984 the Apple Lisa and Macintosh teams released a computer called Lisa which is known as the first successful commercially used computer that uses the concept of windows, panels and mouse-driven control1. Software developers after this date started to focus more on integrating a GUI for their released applications since it offers the user an easy way to interact with the software and also it serves as an attractive mean to help draw the attention of new users. Nowadays, this kind of inputs has developed even further, a new input method that doesn’t rely on a mouse-driven interaction, but on a simple finger touches directly applied on the screen. In this paper I’ll explain the concept and the implementation I’ve developed for creating a GUI that runs on the official Raspberry Touchscreen, at first a comparison of different available technologies to implement the GUI is presented, then a detailed explaination of the Application architecture, and finally the implementaion techniques and methods are layed out to help in showing the link between the arichtecture of the Application and the decisions made regarding the design and implementaiotn followed by the results and outcomes.

**Keywords: GUI, User, Design, Event, Touchscreen**

wikipedia. [1] page History of the graphical user interface.

**Content**

[Abstract 1](#_Toc525482253)

[Content 2](#_Toc525482254)

[List of Figures 3](#_Toc525482255)

[List of Abbreviations 4](#_Toc525482256)

[1 Introduction 5](#_Toc525482257)

[2 Development options 6](#_Toc525482258)

[2.1 TKinter 6](#_Toc525482259)

[2.2 QT 7](#_Toc525482260)

[2.3 Kivy 8](#_Toc525482261)

[2.4 Web App 9](#_Toc525482262)

[3 Application structure 11](#_Toc525482263)

[4 Application parts 12](#_Toc525482264)

[4.1 Main menu 12](#_Toc525482265)

[4.1.1 Turning off the Raspberry Pi 14](#_Toc525482266)

[4.2 Detection Window 15](#_Toc525482267)

[4.3 Speed and Sensors 15](#_Toc525482268)

[4.3.1 Displaying Engine Power 16](#_Toc525482269)

[4.4 ASE-Protocol 17](#_Toc525482270)

[4.5 Overall Module 17](#_Toc525482271)

[5 Sending and Receiving information 18](#_Toc525482272)

[5.1 The implementation of the shared interface 20](#_Toc525482273)

[5.2 Interval value 22](#_Toc525482274)

[6 Conclusion 24](#_Toc525482275)

[Bibliography 25](#_Toc525482276)

**List of Figures**

[Figure 1: Overall structure. 11](#_Toc455056942)

**List of Abbreviations**

|  |  |  |  |
| --- | --- | --- | --- |
| **GUI** | Graphical User Interface |  |  |
| **ECU** | Electronic Control Unit |  |  |
| **HTML** | Hyper Text Markup Language |  |  |
| **CSS** | Cascading Style Sheets |  |  |
| **PHP** | Hypertext Preprocessor |  |  |
| **JSON** | JavaScript Object Notation |  |  |
| **UI** | User Interface |  |  |
| **Tkinter** | TK Interface |  |  |
| **OS** | Operating System |  |  |
| **GCC** | GNU Compiler Collection |  |  |
| **XML** | Extensible Markup Language |  |  |
| **IoT** | Internet of Things |  |  |
| **QML** | QT Modeling Language |  |  |
| **API** | Application Programming Interface |  |  |
| **AJAX** | Asynchronous JavaScript And XML |  |  |

# Introduction

Creating a Graphical User Interface (GUI) is an important part of every application that offers a form of interaction with the user. This part of the software is the only part that the user can see. When not presented well, some users may not be happy about the application on their hands no matter how efficient or fast it is and regardless of all the efforts that were spent creating the underlying parts of the program, after all, one can think of the GUI as the connection point between the user and the device.

There are many tools and technologies available to create interesting GUIs, choosing the right tools and technologies is the first step in creating something with good quality, but unlike other parts of the software, GUIs don’t just need to have good performance and be practical but also, responsive, user-friendly, where it takes into account the age class of its main users base and whether the application will be used by old people, for example, and also attractive since in this time users also expect their applications to have a good design and styling with suitable colors and graphics.

Web technologies were chosen to help in creating the GUI that will run on the official Raspberry Pi Touchscreen, this decision is widely practiced by developers all over the world when creating User Interfaces (UIs) for their applications to run on the Touchscreen. Of course, this is not the only possible option as there are many different tools, libraries and frameworks that can also help developers to create GUIs for the Touchscreen, but this options has some advantages that are not available in the other tools while at the same time offers almost all the main features that are supported by the other technologies.

# Development options

There are many options for developing a Graphical User Interface (GUI) to run on the Raspberry Pi, The search for the tools to be used for implementation was narrowed to the following options:

* TKinter
* Kivy
* QT
* GUI as a Web application (Web technologies)

Those four options are very popular when developing applications for Raspberry Pi that involved graphics, the reasons why those four options are the most popular tools for developing graphics application on this platform are numerous and each tool comes with its advantages and disadvantages.

## TKinter

The name of TKinter is derived from the two words TK and interface since TKinter is a Python binding for the TK GUI framework. The very first thing to notice about TKinter is that it is a Python Library, programs written with TKinter use Python programing language. TKinter comes already pre-installed on the Raspbian Operating System (OS) and the user doesn't need to install any libraries, it is already there and the developer can start focusing on the program on hands right from the very beginning.

TKinter is not dependent on Raspbian, on the contrary it is a platform independent, and the developer can use it to write programs that can be run on many different platforms.  The biggest reason behind the popularity of TKinter is that it is a very fast library, programs developed with it runs smoothly on most hardware without giving an extra attention for performance, this way the programmer can focus more on the functionality that's required to be implemented as a first concern and then the performance since under normal circumstances such programs should run fast.

On the other hand, this library is also known for its old outdated look, usually its look feels a bit old and not suitable for the current changes and improvements happening for GUI frameworks, nevertheless some attempts for changing this reputation took place when issuing TKinter 8.5, unfortunately, these changes were not enough to make programs developed with TKinter look more modern and up-to-date with their appearance. From one side the library is quite efficient in terms of performance but on the other side it doesn't offer the desired look. Offering the user a good and up-to-date GUI is one of the tasks a developer should think about. Offering the look and feel of old GUI styles makes TKinter look unattractive for many developers concerned with creating softwares that has some sort of graphics involved.

## QT

QT is a cross-platform application framework, developed to help in creating programs that need to provide some form of graphical user interface for is users, QT supports many platforms and can run on most Desktop applications, a considerable support for embedded systems as well as Mobile platforms. Programs developed with QT have a natural or native look making it an attractive choice for many programmers, beside the support for GUI, QT can be used to create command-line applications and many interesting projects were developed using QT.

QT also supports a wide range of toolsets and compilers, it supports the famous Linux GCC compiler and Microsoft Visual Studio toolsets, another interesting feature available in QT is that it offers the possibility of writing the logic with JavaScript, QT offers QT Quick that provides what's known as a declarative scripting language QT Modeling Language (QML) which is a declarative language where inline JavaScript is allowed to handle the logic. Moreover, QT offers support for parsing XML documents and processing JSON files, both XML and JSON technologies are widely used in machine-to-machine interactions and webservices hence making QT suitable for applications that need to interact with other machines directly and exchange messages and process them without the need for adding an external tool.

Qt performance has been proven to be good in general although some questions were raised when performing under Raspberry Pi, sometimes the graphics don't render smoothly and some lagging is observed, in such case and especially if the main part of the program that’s being developed is the graphics then the developer might need to make compromises between good performance and good-looking User Interfaces.

Concerning working with Raspberry Pi, QT can offer many desired features as mentioned above but it first needs to be configured and installed properly, one way of installing QT is by creating a cross-compiler for Raspbian, unfortunately, such action is not easily achieved and many developers decide to not consider QT once they fail in creating the cross-compiler, QT online community is already saturated with negative experiences and wasted times trying to setup the cross-compilers to eventually fail after wasting a considerable precious times.

QT isn‘t the best choice, although it offers many interesting features but there are some serious concerns when running the developed graphics on Raspberry Pi, and the fear that using QT for developing the GUI could result in slow unresponsive user interfaces makes QT unattractive for this scenario.

## Kivy

Kivy Is also another Python supported graphics library, it is open source and platform-independent, it can run on Windows, Linux and Android, it is developed by Kivy Organization. One of the strongest point of Kivy is that it supports touch-screens naturally without the need of installing or adding external features, moreover, it supports Raspberry Pi. Kivy is equipped with most of the required features for developing applications, it offers great support for handling user inputs such as mouse, keyboard and as mentioned above for touch events, this becomes quite useful when developing an app for Android. Its graphics are created based on the famous OpenGL library, making it quite flexible and powerful. Kivy also offers what’s known as KV language which is an intermediate language that the developer might need when creating more sophisticated applications.

The downsides of Kivy are not many but they exists, one of them is that Python itself may not be enough to develop a fully functioning application, especially when this application gets bigger, at this point the developer might need to use KV language and the problem with this that KV is only used for Kivy, hence learning it could become costly since time and resources should be spent learning a new programing language and eventually it’ll only be used for creating applications using Kivy.

Although it supports Raspberry Pi, Kivy doesn’t come pre-installed on Raspbian and the developer has to take care of this, the problem here that installing Kivy on Raspbian is not a straightforward process and unfortunately many developers waste long time, sometimes even days just trying to install Kivy on their Raspberry Pi and getting it running to try a simple “Hello, World” program.

Kivy has shown so far that it is a reliable library, programs developed with Kivy showed great degree of efficiency and behaved as expected with the expected response time, while this might be true for small projects, the same can’t be said about big projects. Hence more research needs to be carried on. Kivy still has some work ahead of it when it comes to large-scale projects.

## Web App

The fourth popular option for developing Raspberry Pi applications with GUI is creating the Application as a Web App. This way the application will run inside the browser and can be created with many available technologies such as HTML, CSS, JavaScript, jQuery, and maybe using additional frameworks like Bootstrap, moreover developers also use PHP, Python, or JavaScript (Node.js) when they need back-end technologies, in such cases a web server should be installed, examples for such servers are Node.js or Apache.

This option is very popular among Internet of Things (IoT) applications developers causing it to have a wide online community which means more support for developers. Creating the Application as a Web App, and in this case the GUI, enables the developer to take advantages of the big capabilities offered by HTML and CSS for creating a fast and attractive User Interfaces (UI).

Program created as a Web App will be able to enjoy using all the features available in those mentioned technologies to run fast and smooth, the reason behind this is that, currently websites are accessed from Mobile devices more than traditional Desktop platforms, and some of those Mobile devices are very old running on old software but still perform as expected and are just as responsive.

Web technologies always try to keep compatible with old devices for as long as they can run a browser thus enabling many old slow devices to keep running web apps with acceptable performance and speed and also with attractive UI, such concerns for performance, speed and compatibility make creating applications for Raspberry Pi using web technologies a suitable choice.

The look of the GUI units can be changed and adjusted as much as desired, the reason behind this is that the great flexibility those mentioned technologies offer, when creating a web App the user can expect his app to be accessed from many different devices running on completely different platforms, such challenge was a good point in pushing web development technologies to become more modular and flexible.

Since Web apps can run on most devices equipped with a browser, this means it is even possible to have our application running not just only on Raspberry Pi but also on a Desktop, Mobile devices or even on a Tablet. Such goal could become an important concern for the future and it can be achieved easily, hence it makes developing the Application as a Web App a good choice when thinking about portability features of the App itself.

Graphics created with Web technologies can be developed to become interesting, attractive and user-friendly, such technologies offer many features and possibilities for the developers to create nice, interactive, modular and even animated widgets. One example for such capabilities are the animation feature in CSS, with this feature the developer can create and fine-tone very interesting animated widgets that interacts with the user and respond in an interesting and attractive way to any form of inputs.

Another feature of web technologies is that their ability to handle different forms of user inputs, such as mouse, keyboard and finger touches, while handling multi-touch events is an important feature for Mobile and Tablet apps, it is not an important one when the application running on Raspberry Pi using the official Touchscreen since this screen simulates mouse clicks as finger prints and in this case the application only needs to handle traditional user inputs such as events coming from the mouse or the keyboard. Despite this, this feature could play an important role once there's a need to run the Application on a Mobile device or a Tablet.

One interesting feature when creating web applications that is usually the developer doesn't need to install any additional software or libraries, a mere of a text editor is enough to get the developer ready, on top most of the framework or libraries don't need any installation and can be added by simply adding one line of code to reference them. Such point can save the user hours or even long days of tries to install the right library or bundling different technologies together.

While having many interesting and positive features, web technologies also have their own drawbacks, one of the most obvious one is that any app can only run inside a web browser and if the user doesn’t have an installed web browser then the app can't run, yet this draw back can be overcome, there are currently few available frameworks that helps the developer create applications using web technologies and then producing the final result as a stand-alone app, whether this is for Desktop platforms or for smartphones.

In our case the App will run inside the browser but Raspberry Pi will boot and run the App immediately without any intervention from the user, also the App will be running in full-screen mode, this feature is known as Kiosk mode.

# Application structure

The Application can be developed and structured in many different ways, the connection between the Front-End (GUI) and the Back-End can be established using many different techniques, the used technique is chosen based on performance at first, where the GUI has to show the values sent by the Back-End in real-time then based as well as on other factors such as portability and usability. The options for connecting the GUI with the Back-End include:

* Web socket and a log file.
* Database.
* Using the file system.
* Using JSON technology.

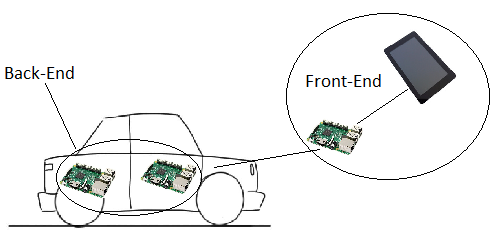


Figure 1: Overall structure

Figure1 shows the overall structure of the System, in this paper the two terms: Front-End and Back-End indicates the following: The Front-End refers to what the user is able to see and interact with, while the Back-End refers to all other Raspberry Pis, ECUs (Electronic Control Units), their connections and software

.

There are also other options that doesn’t include any middleware between the two parts of the System but having no middleware can create some problems and also could lead to a data lose. In this case the Front-End and the Back-End would interact by direct Application Programming Interface (API) calling, those APIs would be an interface that was agreed upon and it represents a contract for interaction between the Front-End and the Back-End.

In this situation there can be two cases:

* The Back-End will signal the Front-End whenever a new data is available to be read.
* The Front-End will periodically call specific APIs to get the desired values and check to see if they are ready to be read.

# Application parts

Internally, the Application depends on many features of HTML/CSS and JavaScript, in the following pages, short description of the most important parts of the application from the point of view of the actual implementation and what kind of technologies were used.

The Application consists of five windows in total. They are divided as follows:

* Main menu
* Detection window
* Speed and Sensors window
* ASE\_Protocol window
* Overall status window

Those five windows collaborate together to offer the user the possibility to monitor the values coming from the different sensors in the car such as the speed of the car, the orientation, the engine power, the sensors on the front and back of the car as well as showing any data detection sent from the Back-End of the System.

## Main menu

When lunching the Application, the first window that will appear is the Main menu, in this window the user can decide one of four options:

* Detection window.
* Speed and Sensor window.
* Overall status of the car window.
* ASE\_Protocol window.

When performing one click over any of them, the icon will get bigger, then a text will replace the icon after few fractions of a second, the way the text replace the icon is carried on as if a curtain starts covering the icon with top-down approach.

To change the size of an item in the menu and make it a little bit bigger, the property “transform” is used as follows:

transform: scale(1.2);

Simply, we are increasing the size of the icon by 20%. Such increase is also timed and happens over 0.2 seconds, this is indicated in the CSS properties of the icon container as follows:

transition: transform .2s;

To achieve the effect of having the text appearing over the icon in a top-down approach, we first have to take care of the text and its container, where we set all the properties depending on how we want the text and its container to appear, such as text color, background color, the size of the text and so on…then we add this effect to the icon whenever the user touches the icon. We also set the timing of such even in the CSS properties of the text container, the effect this time takes place over 0.5 seconds.

Once the user leaves the icon, basically, putting the mouse/finger anywhere outside the icon space, the text and its background will withdraw in a reverse order to the way they were introduced, they’ll move in a bottom-top approach and then the icon will go back to its normal size.

Another option for the user is to choose more than one window, the application offers the user the possibility to combine two windows at the same time so the user can monitor two different reading of the car at the same time. At the very top right corner of the main menu window there are two buttons next to each other, one represents having the screen showing only one module, while the other represents splitting the screen in half and having it showing two modules next to each other depending on user‘s selection. Both buttons can’t be pressed together at the same time, thus the user has to choose one of them, when one option is selected the other option is automatically deselected.

In case the user wants to see only one module, he can go immediately and choose that window, where having only one module showing in the window is the default option. Once a module is selected the application will lunch the selected module. On the other hand, if the user wants to have two modules showing at the same time, the two-module button must be active and it has to be selected first then the user can choose two modules of the available modules. With every module getting selected a green line appears underneath the selected module to indicate the selection, when the second module is selected the application will lunch the new window with having the screen split into two parts and one modules of the previously selected ones appears in each parts of the screen.

The following example shows how the mechanism for the two-option selections. The Application detects which option the user has chosen based on the opacity of each button, since the selected button will have a full opacity while the other will have a 50% opacity.

Once an option is selected, a variable is used to keep the count of how many modules the user selected so far, the reason behind such action is we need to know exactly when the user chooses the second module because once the second module is selected the application will launch a new screen containing both modules:

$("#sas").click(function(){

if( $("#twosr").css('opacity') == '1')

{  
 $("#firsts").css('display','block');  
 number\_selection++;

The above code explains what happens when the user press on “Speed and Sensor” module, when this module is selected the code first checks to see what options of viewing the user has selected, is it one window or tow windows, this is done based on the opacity of each button, if two-window selection is chosen then, at first the application draws a line underneath the “Speed and Sensor” module then increase the counter that keeps track of how many selections.

Later the application checks if this counter equals 2, in this case two modules have already been selected and all the application has to do is to see which two modules those are and then lunches them in one window. Whenever the user makes a selection the application keeps whatever the user has selected in a variable, hence two variables are needed, one for the first module and the other for the second module.

If the user has only selected to see one module each time, then the application will simply lunch a window with the selected module as follows:

window.open('sas.html', '\_self');

The above code, lunches “Speed and Sensor” window, for example, once the user seletcs it after having selected to show the module in full screen.

### Turning off the Raspberry Pi

At the top right corner of the main menu window there’s a small button to turn off/shut down the Raspberry Pi, once this button is double clicked a message will appear to ask the user to confirm the turning off of the Raspberry pi, an answer with yes to the confirmation message will cause the Raspberry Pi to turn off.

This functionality is achieved via PHP, PHP is a server side language and its script can’t be run without installing a server on the Raspberry Pi, so the first step to achieve this functionality is by installing a web server such as Apache, once apache is installed, the next step is to install PHP itself. Both can be installed easily from the terminal.

Now everything is ready and installed, a small PHP script can drive the Raspberry Pi to shutdown via the following line of code:

system('sudo /sbin/shutdown -h now');

Basically, what the function *system* does, is executing an external command/program and outputting the results afterwards, here there are no results. Once this line of code is executed the Raspberry Pi will shut down immediately.

In order to make this code executable/successful, few configurations for the Raspberry Pi has to be made, those configurations can be adjusted from the terminal and for one time only. One of those configurations, for example, is giving Apache (the web server) the permission to shut down the Raspberry Pi.

Rebooting is implemented similarly but with few alternations.

## Detection Window

The point of the detection window is to show a video containing the detected item as well as the sign of the item itself and some information about it. The Detection window uses HTML5 element called “video” to show the video of the detection.

The video element is very simple and can be used with rather few options and properties, the most important one is the path to the video that’s supposed to be played. Besides that, one can specify the width and height for the video element itself using the width and height properties as follows:

<video width="320" height="240" controls>

<source src="movie.mp4" type="video/mp4">

</video>

Here the playing video is called “movie.mp4” located in the very same folder as the source code file, the height is set to 240 pixels and the width is set to 320 pixels.

## Speed and Sensors

Speed and sensor module has three goals:

* Displaying the power of engine, in percentage.
* Displaying the current speed.
* Displaying the sensor data obtained from six sensors located on the front and back of the car.

The Speed and sensor module is divided into two parts, the first part will display the engine power, while the second part displays the current speed and values obtained from the sensors.

### Displaying Engine Power

The displaying of the engine power is done via a circle that has its border as an indicator for the range of the current power of the engine, the way it is shown is by having the border highlighted depending on the actual power range. For example: if the engine is working at 60% of its power, then the circle will has a full border, but only 60% of this border is highlighted and bolded with a different color to the actual color of the boarder. Secondly, the percentage of the engine power is written clearly in the middle of the circle.

Drawing the circle is done mostly in JavaScript, the only thing that needed to be done in html is to have a place holder for the engine power gauge to be drawn at, and this is achieved by using the html canvas element. Canvas by itself can’t draw, it has no functions or properties to help plotting dots or shapes on screen, the developer using Canvas should use as well JavaScript to draw the graphics.

Once we have placed the Canvas element in the html source code, we need a reference to it in the JavaScript function and this is achieved using:

var enginepower = document.getElementById("carengine");

var ctx = enginepower.getContext("2d");

The variable “ctx” has a reference now to the Canvas element in the html document and now it can be used to place graphics in the Canvas. The next thing should be done is clearing the area of drawing, this is achieved using *clearRect* function. The area that’s to be cleared has the shape of a rectangle, so regardless of the shape of the actual drawing the *clearRect* function will clear a rectangular area. Now we can actually focus on drawing the gauge border, we can draw the arc or the boarder of the circle using the *context.arc* function.

Then fill the range of the value that the arc is supposed to show, so far we have built the full border and now we want to draw the value that indicates the current power of the engine, this is achieved again with the same function which is *context.arc.*

## ASE-Protocol

This module consists of three parts:

* 10 LEDs showing status of 10 bits sent by the Back-End.
* Two switches and a slider.
* Four buttons.

The LED buttons will change their statues from glowing to not glowing depending on the respective bits, there are 10 bits that’re set by the Back-End and each bit corresponds to an LED, where a value of 1 means glowing and a value of 0 means not glowing.

The two switches have two values for each one, either 1 or 0 depending on the status of the switch. On the other hand the slider has a value range from 1 to 100.

The four buttons have different behavior, two of them respond to long presses while the other two respond to a normal press.

## Overall Module

This module presents a place for displaying the overall situation of the car, in addition to displaying a chart that helps in having a better understanding of the status of the car. This module could play an important role when trying to debug any malfunctions, damages, or faults happening to the car. Moreover, the module is very easy to adapt and additional information can be added to the car or different types of chart can be presented once required and once the Back-End can present more information regarding the overall situation of the car. This module will show the following:

* Number of total KMs run by the car so far.
* Last time the car was operating.
* Any warning to be displayed.
* A chart which currently showing the working hours each day over a year.

The chart is drawn using canvasJS library, which offers a free trial version.

The code is very adaptable and different libraries can be used depending on the specific requirements at hands.

# Sending and Receiving information

The way the Application deals with data and the messages exchanged between the Back-End and the Front-End is achieved via different types of technologies, those are: PHP, JavaScript, AJAX and JSON. As mentioned above, there are many good advantages of using a shared buffer, and this is the solution that’s being implemented for the System.

At the root directory of the Application, there’s a file called *data.json*, this file contains JSON array that has all the information that’s exchanged between the Application (the Front-End) and the Back-End of the System. The Back-End will write any data that should be sent to this array and the Front-End will present this data to the user.

Having a JSON array as a buffer between the Front-End and the Back-End of the application offers many advantages, the most important one is to have the Front-End completely independent of the Back-End. Whatever technology is used to build the Back-End of the whole System, it can probably work with minimum efforts and interfaces properly with JSON. The job of this array is simple, it’ll basically act as an interface between the two parts of the System, the Back-End and the Front-End.

When new data arises and needs to be presented to the user, all the Back-End has to do is passing this data to the JSON array, this way the Back-End of the System does not have to care about how the Front-End will handle this data.

How can the Front-End know about the new data that just arrived? There are two approaches to solve this problem:

* Having the Back-End signaling the Front-End to indicate that there is new data and the one that’s showing is old and need to be updated.
* Having the Front-End periodically checks for new data and either shows whatever it finds in the JSON array or compares the data in the JSON array with the data it has, if the data is different or even part of it is different, then an update is needed.

Both approaches can, of course, be used, either the first one or the second one, but there are some design principles must be taken into account when deciding which solution is the best one.

Most importantly, the System must present the data in real time, so the focus should be on choosing the fastest option as well. When we design the Front-End in such a way that it needs to be notified to check for any changes in the data, then we are creating another link between the Front-End and the Back-End. Using such design is not a wise decision because we are creating more reasons for reducing the decoupling between the two parts of the System, moreover, such design approach means the designers of the Back-End need to think of a way to send messages and notifications to the other part of the System, that’s the Front-End, and this part must be designed to accommodate and receive those signals, thus both designers must agree on a way to design the System although the Front-End is being designed separately and there’s no direct connection or knowledge of the how the System is going to send messages to the Front-End.

Choosing this option, means the Front-End is going to suggest a way of how it wants to receive signals coming from the other part of the System, whatever technology is being used for this purpose it may not be the right one or the most practical one for the Back-End. Furthermore, there’s the problem of presenting the data to the user in real time. Right now, the Front-End can’t tell which method is going to be used by the Back-End and thus it can’t guarantee that it’ll be able to show data in real-time to the user.

The other option of showing the data coming from the Back-End is having the Front-End itself to do periodic checks to bring data from JSON array and present it to the user. With this option there are also two options to choose from: When reading new data from JSON array, should the Front-End compare the new data with the data that’s already presented and only shows this data when there’s a difference between the two? Or should the Front-End simply show the data that it has just acquired from the JSON array without doing any kind of comparison?

When the Front-End needs to compare the newly read data with the old one the following steps will be followed:

1. Read the data present in the JSON array each X amount of time
2. Read the data that’s currently presented to the user (the old data), since the Front-End doesn’t have a storage mechanism
3. Make a comparison between the two to check for any changes
4. Two cases:
   * + - There’s no change in data, do nothing
       - There’s a change in data between the two readings and the new changes will be presented to the user

On the other hand, When the Front-End simply reads the data available in the JSON array and present it back to the user, the following steps will be followed:

1. Read the data present in the JSON array each X amount of time.
2. Present this newly read data back to the user.

Both options mean the Front-End has to, first, read the JSON array and then reaches the parts of the program that’s showing data to the user. But when using the first option, the Front-End is not done yet and it has to do few more operations before the data is ready to be presented to the user. While when going with the second option, the whole operation is done in two parts: Reading the JSON array, pushing those readings to the user.

The solution to be used is the later one, where the Front-End has to do the minimal possible operations, so it adheres to the real-time requirements as close as possible.

When using such solution, the user will not notice any difference when the Front-End pushes the newly read data in case this data is not different from the old one, the user will notice changes only when the newly read data is different from the data that was showing previously.

## The implementation of the shared interface

The way to access JSON array is achieved via PHP, at first the PHP script must know where the array is stored, so it needs the path of the array, then JSON content of the file must be decoded, and finally each element in the array can be accessed through its key. As mentioned above JSON array is stored in a file called *data.json* and since JavaScript can’t reach the file system, then a different technology must be used, in this case PHP was chosen.

The reasons behind such decision are because PHP is a Back-End technology and thus is able to reach the file system, PHP has simple and very user-friendly methods for accessing and handling JSON arrays, and PHP is already being used in the Application when turning off/rebooting the Raspberry Pi, thus using PHP means there’s no need to introduce new technologies to the Application and avoid any complications that could results with such action. The following code illustrates the process:

$url = 'data.json';

$data = file\_get\_contents($url);

$readings = json\_decode($data);

*file\_get\_contents* is the function that will locate the file containing the JSON array, then *json\_decode* will extracts the JSON array from the file. Now for example if we would like to access the speed element, we can do it via the following:

$readings[0]->engine;

To show the speed of the car in the info bar, we need to call this PHP script that handles the JSON array and then redirect the returned value to the HTML file that’s showing the info bar.

To do this, we first need to specify which part of the HTML document is responsible for presenting the speed, once this is done, we call the PHP script which is returning the speed that’s being read from the JSON array. The calling of the PHP script is being indirect and through another function called *setInterval*, this function will keep calling the function that’s passed to it as its first parameter, in this case the calling of the PHP script, and will keep repeating the calling according to the interval that’s passed to it as its second parameter, the second parameter in this case is *x amount of time* this value is measured in milliseconds and will have the function *setInterval* to repeat the calling of its first parameter every *x amount of time*.

Getting the orientation data of the car from JSON array is done a little bit differently, the reason for this is with the speed, the called PHP script could simply print the speed directly into the responsible HTML file, while with the orientation, we want the PHP script to return the read value from JSON array and JavaScript will have to alter the CSS attributes of the parts of the HTML that’s showing the orientation of the car, since we need to rotate the arrow that points to the direction of the car and this rotation can be done with CSS.

The function *setInterval* is used again, with two parameters, the first is the AJAX script that will call the PHP script which will return the orientation value from the JSON array, and the second parameter is the *x amount of time*, meaning that the function *setInterval* will call the included AJAX script every x amount of time. The difference is in the AJAX script, as shown in the following code snippet:

$.ajax({

type: "GET",

url: 'orientation.php',

success: function(data){

$(".orientation").css("transform", "rotate("+data+"deg)");

}

})

Here AJAX script is making a GET request to the PHP server that’s running the PHP script, the request is directed to the page *orientation.php* and the returned value after calling the PHP is passed to the variable *data,* the code that’s inside the function will be executed once the PHP script return with a status of success from its call.

Since the orientation value that’s read from the JSON array is now stored in the variable called *data* we can now use this value inside the function that’s called in the AJAX script to alter the HTML part that’s showing the orientation of the car according to the value *data*.

Since the arrow that’s showing the orientation of the car can be modified using CSS, we can simply alter its CSS properties accordingly, we do this using jQuery. First the arrow must be located, using *$(“.orientation”)* then we specify which attribute we want to alter, and then we pass the value, in this case we are modifying the transform property, and applying a rotation with a *data* (data is the orientation value obtained from the JSON array) degree.

## Interval value

What is the right interval that should be used to make a call for the PHP scripts to obtain the required values from the JSON array? The answer to this question can partially be derived from experience, among other factors, where this value should not be set arbitrarily.

When the Application doesn’t make enough calls to obtain the new values then the Application is missing many important values and not doing a good job of showing data in real-time to the user. On the other hand, when the Application is doing many additional unnecessary calls, it is then overwhelming itself which will lead to many performance issues that could make the experience of the user unpleasant.

Fortunately, this value can easily be changed and set, it is hardcoded because there’s no reason to set such a sensitive value as a choice for the user. Having the value as an option where the user can enter this value in a textbox, for example, could put the Application under many risks, since the user doesn’t have any idea about the underlying of the code and the architecture of the Application.

The nature of the Application allows for great performance even when this value is set very low, the reason behind this is there are no threads running in the background, and the only running part of the program is what the user can see on the screen, so at any given time there will be only one page loaded and running and the Application will be trying to make calls for the JSON array only for the data related to this specific page, any data that’s presented in other pages and needs to be obtained from the JSON array will not be called unless it is in the active page itself.

Another important point to be noticed here, is that this interval value can be different from one part of the Application to another, where some values that’s sent from the Back-End and saved into the JSON array don’t have a sensitive nature where it can change every few milliseconds of time, thus the interval between each call for such values shouldn’t be very small and should be set according to the nature of the obtained values/date.

Besides taking care of presenting the values in real-time, the interval values should not be too small to cause the application to make unnecessary calls and access the JSON array without an actual need, such behavior could result in bad consequences and cause the Application to show some performance issues like being less responsive, or slow when responding to the user’s interactions, another issue with having small interval values that could lead to making the Application not able to obtain other values from JSON array in real-time and hence having a backward consequence while trying to make the Application sensitive for real-time reading.

On top, setting some values too small, could actually not have the benefit of having the user reading real-time values because when having the interval too small and the values are changing at a very quick rate such as every 5 milliseconds and the presented values are changing at a similar rate, the user will not be able to notice and read all the changes because the human eye have a threshold and can’t see very quick changes, so when choosing an interval value, we should also take into account the user’s ability to read and at which rate is also acceptable for the user because the point of those values is to be presented to the user but not to be used by a machine or a software which unlike humans can notice every small change and make use of it.

# Conclusion

The Application that’s has been implemented serves its goals and purposes, it is responsive and offers the user a good experience. One of the strongest points of implementing the GUI using web technologies is to have the application easily ported to any platform, after all, the only requirement for the host device is to have a web browser. In future, when the Application is needed to run on a PC (Windows or Linux), Mac, IPhone, an Android device, or an IPad it can easily run.

Currently, the Application components and UI is designed to look good on the Official Raspberry Touchscreen, and they’ll not adjust well for any different screen size but once needed it can be ported to look well on all the mentioned devices with not much efforts. One reason for not having the Application to automatically adjust to any screen size as it is now is having the focus on creating something suitable for the Touchscreen first and for most, as having it to run on every device is not a current requirement.

On the other side, the provided Touchscreen simulates mouse clicks when getting touch inputs, this approach doesn’t work well on the browser which causes some unwanted behavior sometimes for example when the Touchscreen mistakes a swipe with a long press assuming that a long press from the user is intended to be a swipe, hence extra work needed to capture such misunderstanding and take measures to eliminate unwanted behavior.

Bibliography

[1] Raspberry. (2018) Raspberry pi [Online]. Available:https://www.raspberrypi.org. [Accessed May 10, 2018]

[2] Mark Summerfield, *Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming*. Prentice Hall, 2015

[3] Boudewijn Rempt, *GUI Programming with Python: QT.* Opendocs Llc, 2002

[4] Shaik Riyaz Hussain, K. Rama Naidu and Chintala R. S. Lokesh, “2D-game development using Raspberry Pi” in *Information Communication and Embedded Systems (ICICES), 2016 International Conference,* 2016

[5] Ahmed Bader, Hakim Ghazzai and Abdullah Kadri “Front-end intelligence for large-scale application-oriented internet-of-things,” in The Plethora of Research in Internet of Things (IoT), 2016, pp. 3257 – 3272

[6] Peipei Ling, “Based on web application front-end performance optimization” in Electronic and Mechanical Engineering and Information Technology (EMEIT), 2011 International Conference, 2011

[7] Gerd Wagner, “Introduction to simulation using JavaScript,” in Winter Simulation Conference (WSC), 2017

[8] Soham Kamani, *Full Stack Web Development with Raspberry Pi 3: Build complex web applications with a portable computer.* Packt Publishing *, 2017*

[9] Stephen Chin and James Weaver, *Raspberry Pi with Java: Programming the Internet of Things (IoT) (Oracle Press).* McGraw-Hill Education, 2015

[10] Derek Molloy, *Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux.* Wiley, 2016

[11] Ali Sajedi Badashian, Mehregan Mahdavi and Amir Pourshirmohammadi “Fundamental Usability Guidelines for User Interface Design” in Fundamental Usability Guidelines for User Interface Design, 2008

[12] Heshan Liu, Fujun Ma “Research on visual elements of Web UI design” in IEEE 11th International Conference on Computer-Aided Industrial Design & Conceptual Design 1, 2010

[13] IconFinder. (2018) IconFinder Dashboard, diagram, gauge, graph, infographic, meter, statistics icon. Available: https://www.iconfinder.com/icons . [Accessed Jun. 17, 2018]

[14] Adream Blair-Early, Mike Zender, “User Interface Design Principles for Interaction Design,” in Design Issues, 2008, pp. 85 – 107

[15] Sadeque Reza Khan, M. S. Bhat, “GUI based industrial monitoring and control system,” in Power AND Energy Systems: towards sustainable energy, 2014

[16] Caroline Labib, Ehab Hasanein, Osman Hegazy “Early development of Graphical User Interface (GUI) in Agile methodoligies” in The 7th International Conference on Informatics and Systems (INFOS), 2010

[17] Rodrigo M. L. M. Moreira, Ana C. R. Paiva, Atif Memon, “A pattern-based approach for GUI modeling and testing,” in IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), 2014

[18] Ali Sajedi Badashian, Mehregan Mahdavi, Amir Pourshirmohammadi, “Fundamental Usability Guidelines for User Interface Design,” in International Conference on Computational Sciences and Its Applications, 2008

[19] Lu Xudong, Wan Jiancheng, “User Interface Design Model,” in Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007

[20] Muhammad Umar Hayat, Nauman Qadeer, “Intra Component GUI Test Case Generation Technique” in International Conference on Information and Emerging Technologies 2007

[21] Rodrigo M. L. M. Moreira,Ana C. R. Paiva, Atif Memon, “A pattern-based approach for GUI modeling and testing,” in IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), 2013