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

* TKinter
* Kivy
* QT
* Create the GUI as a Web application.

Those four options are very popular when developing applications for Raspberry Pi that involved graphics either as part of the application itself or as part of the GUI, the reasons why those four options are the most popular tools for developing graphics application on this platform are numerous and each tool come with its advantages and disadvantages.

TKinter

The very first thing noticed about TKinter is that it is a Python Library, programs written with TKinter use Python programing language. TKinter come already pre-installed on Raspbian and the user doesn't need to install any libraries, do any kind of configuration or preparing any tools, it is already there and the developer can start focusing on the program on hands right from the very beginning.

The name of TKinter is derived from the two words TK and interface since TKinter is a Python binding for the TK GUI framework.

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 user 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 program 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On the other hand, this library is also known for its old outdated look, usually its look feels a bit old and not suitable for the current changes and improvements happening for GUI frameworks, nevertheless some attempts for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 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, it is not the most important concern though since the developer should concern himself with the main functionality of the program that's being developed. Providing a bit of an old appearance GUI makes TKinter look unattractive for many programs concerned about creating programs 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, 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 QML which allows the developer to use JavaScript for creating the logic of his program.

Other important features offered by QT is that it provides developers with platform-independent Databases API where the developer can have access to many different Databases Managements Systems (DBMS) regardless of the current platform, some of the supported DBMS include:

* MYSQLDBMS using QMYSQL.
* MS Access DB and SQL Server as well as other ODBC- compatible Databases via QODBC.
* PostgreSQL using QPSQL.
* Different versions of SQLITE through QSQLITE
* Supports Oracle-based Databases via QOCI.

Moreover, QT offers support for parsing XML documents and processing JSON files, both XM and JSON technologies are widely used in machine-to-machine interactions and webservices making QT suitable for applications that needs to interact with other machines directly and exchange messages, processing them without the need for adding an external tool.

Even when applications need to be scaled and the program being developed on hand is getting bigger and requires some type of threads processing to handle different data processing and different operations simultaneously, QT can still be a possible choice since it offers a good support for threading. Just like the supported Databases API, QT support for threads is also platform-independent and the developer has a wide range of classes and possibilities when it comes to writing multi-threaded applications. Further, QT also supports working on multi-core processors thus the program can take advantage of the possibilities and powers offered by the hardware.

Qt performance have been proven to be good in general although some complains were raised when performing for 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.

QT is not all good and strong points, as with any other tool it has its own week points and drawbacks, 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 and developing programs using its framework 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.

For developing the GUI-main unite part of the project, QT doesn't seem to be 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 may create problems and additional overhead since QT would be the right choice for larger projects.

Web App

The fourth popular option for developing Raspberry Pi applications and the ones that needs GUI support is by 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 framework 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 on the Raspberry Pi in case the developer wants the back-end part of the App to keep running on the same Raspberry Pi, 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 meaning more support for other developers when an issue arises while developing the App.

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)

Such program created as a Web Ap 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 the perform as expected and are just as responsive.

Web technologies always try to keep compatible with old devices for as long as possible thus enabling many old slow devices to keep running web apps with acceptable performance and speed and also with attractive UI, such concern for performance, speed and compatibility makes 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 enjoy, since 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 making it modular and a flexible choice.

Web App now are accessed all over the word 24/7, using very different devices, some of them have great powers and perform very fast while other devices are old, slow and still using old technologies that were developed years or even over a decade ago, this is one of the most important features of Web Apps, any Web App is able to run on any device as long as this device can handle and run a web browser, the Web App will be lunched from the browser and then will live and die inside the browser, if the user's device can open and run a web browser it is highly possible that the App that's being developed with Web technologies will run and perform well on any device with a web browser.

This platform-independent approach is very important, and in such case the developer can now focus on the App itself and its main requirements and functionalities and not to worry much about the users base and what device they may or may not have.

In the case of our Application, this point makes it 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 one for the future and it can be achieved easily, hence making developing our application as a Web App a good choice when thinking about portability features of the App itself.

Web development sector enjoys a wide online community and usually the developer can find answers for is questions, solutions for common issues and clarifications of why the App is behaving this way or why it is not behaving the way it is supposed to.

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 responds in an interesting way to any form of inputs.

Another interesting feature of web technologies is that their ability to handling 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 usually the user 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 various Desktop platforms or for Mobiles and 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 and will be explained shortly.

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 its graphics quite flexible and powerful in what can be achieved with them.

Kivy also offers what’s known as KV language which is an intermediate language the developer might need when creating more sophisticated applications.

Graphics created with Kivy have an up-to-date look and feel, thus making it quite attractive when developing applications that requires some form of GUI.

The downsides of Kivy are 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 expensive since time and resources should be spent to learn the new programing language and eventually it’ll only be used when creating applications using Kivy.

Although it supports Raspberry Pi, Kivy doesn’t come pre-installed on Raspbian operating system 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 is still relatively new library and its internet community is growing but once a developer gets stuck at an issue and try to seek help from fellow developers online, he may not find someone ready to guide him, since Kivy’s community is still growing but not big enough, the same goes for resources online, learning the basics of Kivy is considered easy and many materials, tutorials and websites offer such help, but once the developer needs to mover further from the basics into more advanced stuff, acquiring such knowledge becomes a big problem because of the lack of online resources for advanced topics.

Kivy has shown so far that it is a reliable library where 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 developed using Kivy and still much more research needs to be carried on. Kivy still has some work ahead of it when it comes to larg-scale projects.

The mentioned downsides of Kivy made it hard to choose it as the tool of choice for creating and developing the GUI and it’s elements, moreover one of Kivy’s great advantages is that its support for touching events coming from the user, while this can be a great benefit and a life-saver for android developers but it doesn’t make a difference and doesn’t put Kivy ahead of other tools when developing applications that target the official Raspberry Pi touchscreen, since the screen itself simulates user touches as mouse clicks, meaning when creating applications for this touchscreen the developer can simply use mouse events and mouse clicks as users finger touches.

Overall structure of the Application

The Application can be developed and structured in many different way and 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, where the GUI has to show the values sent by the Back-End in real-time. Those options for connecting the GUI with the Back-End include:

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

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

In this situation there can be two cases:

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

If no middleware or some form of buffer is used then some problems might arise, when using some form of a common or shared buffer and the Application has to be fine-toned and designed according to the rules of multi-process synchronization. The reason behind such consideration is that the Backend is probably going to produce many data in short amounts of time, since the car relays in many sources of inputs and also needs to draw the attention of the driver to various data readings.

Dealing with such great amount of data and having the reader-writer or producer-consumer model means that the whole system must be synchronized and the data must be consumed or read at the right time by the reader to avoid any loss of data, since the buffer could have its limit and at one point the writer might not find an empty slot to place its newly generated data.

Let’s consider the following scenario where no shared buffer is used:

Assuming where are talking about adjusting the speed of the car, here we have a producer-consumer model and we have an API that the Frontend can call which in turn will make a call to the Backend, the Backend will process the request and send the result back which will be read by the Frontend.

The Frontend makes a request to read the speed of the car, the request goes to the Backend which in turn brings the result back to the Frontend, in our case the data is the current speed of the car, during this process the Frontend made the initial request because the driver was changing the speed of the car and thus a reading of the current speed must be provided, assuming the user kept changing the speed of the car while the whole request process was happening then there’s a significant risk that the speed that’s finally delivered to the driver is not the current speed of the car, where the actual speed might have been altered while the data was being sent from the Backend to the Frontend.

When considering the solution which includes a common buffer, the designers of the solution must consider the problems associated with this model, those problems include, but not limited to:

* Race condition.
* Deadlock.
* Overflow of the buffer.

Race condition between the Backend and the Frontend may not be a problem here since one only reads data and the other writes the data, both parts don’t have the capabilities of reading and writing at the same time meaning race condition can be avoided in principle.

Deadlock could happen when the producer, the Backend, is waiting for a signal from the consumer, the Frontend, to write the data it has produced to the buffer while at the same time the consumer itself is waiting for a signal from the producer to read the data that it is waiting for. In such scenario both parts of the Application are waiting for each other and they could end up waiting for ever.

The third risk that should be taken into account is having an overflow of the buffer. At some point the consumer may not be able to consume data produced by the producer at the same speed as it is being produced, and in such case the car is still reading values coming sent from different sensors and after processing them the Backend may not be able to write them all or some of them to the shared buffer, because the buffer is full now and it can’t accommodate more data. This means that the reader will not be reading a real-time values since some of the values that exist currently in the buffer are old and don’t represent the current state of the car.

Here the producer will keep waiting until a new place is empty in the buffer, and while waiting new data also could be produced making the problem even bigger.

Solving this problem can be achieved by designing a buffer that’s big enough where it can accommodate as much data as possible even if the consumer is having troubles and or is a little bit behind the producer. On the contrary of other applications, such problems here are not accepted, even if the buffer is big enough and can accommodate most of the data being produced by the producer, it is still not acceptable that the Frontend is not processing that data at the required speed and that its readings are not the current readings.

In this scenario our Application is critical and incase of problems of the delay or any other types of issues, it could lead to serious risks that may result in catastrophic consequences. Not showing the actual speed of the car, for example, could misguide the driver and will most likely affect his decisions of speeding or reducing the speed of the car which could lead to accidents in the worst case.

Linking the Backend and the Frontend of the Application is not a matter of passing data from one application to the other, it is a design concern and big factor in making the care more stable, reliable and even safer.

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 providing a simple controlling mechanism for few function of the car..

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

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

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 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.

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

$('#onesr').css('opacity','0.5');

$('#twosr').css('opacity','1');

});

The above jQuery code snippet shows how the alternation between activating one window or two window, “twosr” is a reference to the two-window split button, while “onesr” is a reference to the one-window button, when the user presses one splitting the screen then, the two-window selection button is activated and it has a full opacity while the other option which is one screen is deactivated and has a 50% opacity.

In case the user wants to choose having only one module in the window, he can go immediately and choose that window, where having only one module showing in the window is the default option, thus when lunching the application and if the user wants to see only one module at a time, they can go select that module directly. 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 appear in each part of the screen.

The following example show how the mechanism for the two-option selections and the modularity of the application:

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:

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

Once we are inside the curly brackets, it means the user has already chosen two-module selection and now we should detect which are the two modules and move on to viewing them.

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 selection 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 lunching them in one window.

Whenever the user makes a selection the application keeps whatever the user has selected in a variable, thus two variables are needed, one for the first module and the other for the second module.

All the code snippet above is about showing the two modules in the same window based on the user selection.

On the other hand 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.

Main menu

The menu has the role of directing the user to the desired window, the user can choose one of four options:

* Speed and Sensors.
* Detection.
* Overall Status.
* ASE\_Protocol.

When hovering 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 done as if a curtain starts covering the icon with top-down approach.

To change the size of an item in the menu and make 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 curtain 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 the text color, the background color, the size of the text and so on…then we add this effect to the icon whenever the user hover on 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.

transition: .5s ease;

The ease property is added to indicate that the animation speed will not be all the same during the whole time, it’ll be slow at the beginning, then it will go faster and then it’ll be slow at the end of the animation.

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.

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 lay 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, once 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 in 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 first of all 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, as follows:

<canvas id="carengine " width="300" height="300"> </canvas>

The code above, simply, represents a container for the engine power gauge to be drawn at, it has an id property so the element can be identified in the Cascading Style Sheet and can be styled properly, it also has a width and height values to control the width and height of this container respectively.

The actual drawing is achieved through JavaScript.

Canvas element is used to draw graphics using, mostly, JavaScript, having said that, 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. This function takes four parameters:

* x: Represents the X coordinates of the top right corner of the rectangle.
* y: Represnts the Y coordinates of the top right corner of the rectangle.
* width: Represents the width of the rectangle area.
* height: Represnts the hight of the rectangle area.

Once this is done, we can actually now focus on drawing the gauge border, we can draw the arc or the boarder of the circle using the *context.arc* function. This function takes many parameters as follows:

context.arc(x,y,r,sAngle,eAngle,counterclockwise)

* X: Refers to the X coordinates of the center of the circle.
* Y: Refers to the y coordinates of the center of the circle.
* R: Refers to the radius of the circle.
* sAngle: Refers to the starting angle of the drawing.
* eAngle: Refers to the end angle of the drawing.
* Counterclockwise: Refers to the direction the drawing should follow to draw the arc, this can take one of two values, either “true” or “false”. The “True” value indicates the drawing should go counter-clockwise, while the “false” indicates the drawing should go clockwise, and this is the default value.

Now we can actually 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 to the current power of the engine, this is achieved again with the same function which is context.arc

Once we are done with drawing the range of the value we can also set the properties of the new drawing, we need to make the color of the range of the engine power different from the first boarder we drew, and we can set the color using:

ctx.strokeStyle = color;

Same also goes for the width of the line, we want the width of the range value to be thinner than the thickness of the actual boarder of the gauge, this can be done using:

ctx.lineWidth = *value*;

Once all the preferences are set and all the required values are in place we can go ahead and draw the range of the values using:

ctx.stroke();

To write the percentage value of the power of the engine in the middle of the gauge we have to use a function called *fillText*, this function has the following signature

ctx.fillText(text,x,y,maxWidth);

* text: is the text we are writing.
* x: is the starting X coordinates.
* Y: is the starting Y coordinates.
* maxWidth: the maximum possible width of the text, this parameter is optional.

Turning off the Raspberry Pi

At the top right corner of every screen 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 logical 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 so nothing will be output.

Once this line of code is executed the Raspberry Pi will shut down immediately.

In order to make this code executable/successful, very configuration for the Raspberry Pi has to be made, those configuration can be adjusted from the terminal and for one time only.

One of those configuration for example is giving Apache (the web server) the permission to shut down the Raspberry Pi, this is achieved by writing this permission to the sudoers file, and the permission is as follows:

www-data ALL = NOPASSWD: /sbin/shutdown

The above line will give Apache the rights to shut down the Raspberry Pi without asking for a password.

Rebooting the Raspberry Pi

At the top right corner of every screen and next to the shutdown button there’s another button to reboot the Raspberry Pi, once this button is double clicked it’ll show a confirmation message to ask the user of whether they actually want to reboot the System or not, once the user confirms the rebooting, the Raspberry Pi will reboot.

Just like with turning off the Raspberry Pi, rebooting is also achieved via PHP using the following script:

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

The above script once called will cause the Raspberry Pi to reboot, also Apache must have the permission to order a rebooting of the System. This permission can be given using the following line:

www-data ALL = NOPASSWD: /sbin/reboot

Sending and Receiving information

The way the Application deals with data and sending and receiving information 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 Application.

At the root directory of the Application, there’s a file called data.json, this file contains JSON array that has all the information which 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 and view to this array and the Application 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, one of 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 interface properly with JSON.

The solution that’s has been selected is having a shared buffer between the Front-End and the Back-End of the System, the buffer job 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 need 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 those data. How would 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 and strategies 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 are creating another link between the Front-End and the Back-End of both parts of the System. 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 sending 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, this problem. 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 a periodic check 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 read data with the data that’s already presented and only show this data when a there’s a difference between the two? Or should the Front-End simply show the data that it has just acquired from the JSON 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 then it has to reach 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 solution, 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 underling implementation of the used solutions:

The way to access JSON array is achieved via PHP, at first 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 of 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. This is done in the JavaScript file that’s related to the info bar.

var container = $("#carspeed");

var refreshId = setInterval(function()

{

container.load("speed.php");

}, 100);

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 for the PHP script, and will keep repeating the calling according to the interval that’s passed to it as its second parameter.