Implementation Guide

[Intro 2](#_Toc513476786)

[Hardware 2](#_Toc513476787)

[Basic Car Functions 2](#_Toc513476788)

[Handshake/Engine Start 2](#_Toc513476789)

[Back Button Exit App/Engine Stop 3](#_Toc513476790)

[Motor Control 4](#_Toc513476791)

[Home Button Pause App/No client/Parking Brake 4](#_Toc513476792)

[Handbrake 6](#_Toc513476793)

[Throttle/Brake/Neutral 7](#_Toc513476794)

[Reverse 9](#_Toc513476795)

[Cruise Control 9](#_Toc513476796)

[Steering Left/Right/Neutral 10](#_Toc513476797)

[Electric Car Functions 12](#_Toc513476798)

[Lights Front (Low / Normal / High) and Back 12](#_Toc513476799)

[Brake Lights (Back) 12](#_Toc513476800)

[Turn Lights (Left / Right / Straight) 12](#_Toc513476801)

[Emergency Lights 14](#_Toc513476802)

# Intro

This is an implementation guide for connections between server (Raspi) and client (Android app). Rarely, during development I changed my mind about implementation and I changed the way of thinking. Guess what, then I was bored to update this file. Read the source if you want to see the way I implemented everything.

# Hardware

Make sure that you do not waste any pin that has extra functionality (i2c, spi, etc).

The LEDs will be controlled through Shift Register.

SPI will give me more (16) pins for extra work.

I2C will have sensors (gyroscope, etc) through a multiplexer.

# Basic Car Functions

## Handshake/Engine Start

### Client

* 1. When the user long-clicks the engine start-n-stop (ImageView), a pop-up dialog will appear with an EditText in which the user enters server IP and port (server\_ip:port). At the bottom of the dialog there will be an “OK” button. When the user presses “OK” button the client will try contact the server “server\_ip:port/start\_engine(nanohttp\_client\_ip, nanohttp\_client\_port)”. The server will respond back with a message (“OK”). After a successful handshake the engine start-n-stop will change state icon but if the handshake was unsuccessful a toast message will inform the user about the error. The “server\_ip” and “server\_port” will be update from null and -1 to the correct values.

### Server (DONE)

* 1. The server will listen for calls “EngineSystem.handshake(nanohttp\_client\_ip, nanohttp\_client\_port)” from clients. In case this method doesn’t receive any arguments the default will be “nanoClientIP = null” and nanoClientPort = -1.
  2. Server will call hardware method “car.controllers.basic.EngineImpl.start()” to initialize GPIOs,PWM, etc .
  3. *TODO it must return if the engine (GPIOs, PWM, etc) has successfully been initialized.*
  4. *~~POSSIBLE SOLUTION 1:~~* ~~I will use Kotlin coroutines (Future-Callable?) in order to wait for a while for "EngineImpl.start()” to finish and get it’s result (Pair). The “handshake(...)” method will return the String part of the Pair.~~ ‎
  5. *POSSIBLE SOLUTION 2:* Why to use coroutines for a, by default, sequentially execution? The “car.controllers.basic.EngineImpl.start()” will be called sequentially by default!

### Hardware (ALMOST)

* 1. There will be an interface “car.controllers.basic.Engine” with 2 methods “Pair start()” and “Pair stop()”. Singleton “car.controllers.basic.EngineImpl” will inherit from “Engine” will be singleton class. Here will discuss “start()” method.
  2. *How about instead of functions “start()/stop()” to replace them with a “boolean var”?*
  3. *Maybe it would be better if “start” and “stop” methods have the working source at the interface. It would be more extendable when I would like to initialiaze more GPIOs.*
  4. “Start()” method will initialize every pin I need (set mode = input/output, set every pin to 0, etc) and before exiting it will set “engineState = true” (“true” means started, “false” means stopped). Then it will return a Pair<String, Boolean>(Pair(“OK” or error\_message , engineState)).

### Extras (DONE)

* 1. It also needs a function in the client which asks the server which asks the hardware if the engine has successfully started/stopped. The client will have a “val isEngineStarted” with a modified “get” function. This will call a server function “get\_engine\_state()”. This function will be in the “EngineSystem” class. This server function will read the state of the “engineState” from the “Engine/EngineImpl” class.

## Back Button Exit App/Engine Stop

### Client (DONE)

* 1. This action will be activated when the user long-clicks the engine stop ImageView or presses the device back button twice.
  2. This action will executes client’s function “stopEngine()”. This method will call server’s “stop\_engine()” function and will wait for a termination signal (like in “startEngine(...)”).
  3. After a successful termination (“OK” signal) the engine start-n-stop ImageView should change icon to “engine stopped” and if the initial action was triggered by the device’s back button the app should also “finish()”

### Server (DONE)

* 1. The server’s function “EngineSystem.stopEngine()” will call “ EngineImpl.stop(): String”.

### Hardware

The function “EngineImpl.stop()” will shutdown, unprovision the GPIOS, PWM, etc and whatever it needs in order to have a bugless engine start after this shutdown.

* 1. *How about instead of functions “start()/stop()” to replace them with a “boolean var”?*

## Motor Control

The values which are sent and applied must be tested. The values applied are not correct at the moment because for example 0/100 values at parking brake and handbrake should not be applied at the real world. 0 will be fast brake and 100 will be full throttle, this means when I release the parking brake full throttle will be applied, which obviously is not correct. I have to find the perfect value in which the car is in neutral.

## Home Button Pause App/No client/Parking Brake

### Client (DONE)

* 1. Parking brake will be activated with long-click at the ImageView. Parking brake will be deactivated the same way. Handbrake is another ImageView and it will not be discussed here.
  2. When the user long-clicks this ImageView the “activateParkingBrake(state=true/false)” at the “.network.ClientTriggeredRequests” will call appropriate server’s function “ThrottleBrakeSystem .set\_throttle\_brake\_system(id=incremental\_int, action=ACTION\_PARKING\_BRAKE, value=0/100)”.
  3. Get function of the “isParkingBrakeActive: Boolean”will call server’s “ThrottleBrakeSystem.get\_parking\_brake\_state()” and will wait (using blocking coroutines) for a result from the server function.

### Server (Almost)

* 1. Server’s function “ThrottleBrakeSystem.set\_throttle\_brake\_system(id=incremental\_int, action=ACTION\_PARKING\_BRAKE, value=0/100)” will call hardware’s “ThrottleBrakeImpl.parkingBrake(value=0/100)” function. Server’s function will return a String.
  2. *At the moment the following is based on server events and not from hardware events.*
  3. Server’s function “ThrottleBrakeSystem.get\_parking\_brake\_state” will call hardware’s “var ThrottleBrakeImpl.isParkingBrakeActive”with modified “get” function. Server’s function will return a Boolean.

### Hardware

* 1. Parking brake is different from handbrake. Parking brake applies to 4 wheels (if the vehicle is awd) and handbrake applies to the rear wheels.
  2. As said at server section, there will be a “ThrottleBrakeImpl.parkingBrake(value=0/100)” function which will apply the parking brake. The return String of this function should be the value of the pin, but if I cannot get this value it will return 0 or 100 according to the function argument or an error message according to the exception. Before returning the String value (not the exception message) and if and only if I could not get values from PWM pins, do not forget to “set” the appropriate value to the “var ThrottleBrakeImpl.isParkingBrakeActive“.
  3. The “get” function of the “var ThrottleBrakeImpl.isParkingBrakeActive“ will be modified also, because I think that I can get data back from PWM pins. If I cannot get data back from PWM pins then there will be a regular “get” function. This “var” will have a private “set” function.
  4. “ThrottleBrakeImpl” class should have a private function “applyPwmValues(motorXXPinYY, …) where XX is FR, FL, RR, RL and YY is 00, 01, 11, 12”. I know that the “enable” pin is missing from this setup but it will be added in during implementation. When this function is called using all arguments every time it will automatically unset the previous settings from the motors.

### Extras (Almost)

* 1. The server should scan repeatedly if the client is online, otherwise the parking brake must be applied because the car is moving without control. At the early stages of the project it will be used “InetAddress isReachable(..)”, but later it will check for the nanohttp server in mobile. This will be done because “InetAddress isReachable” is not reliable according to some guys at the internet (and there are a lot of guys saying it, so it must be true!)

### ~~Advice~~

* 1. ~~If I could get values from PWM pins the hardware part could remove the function “ThrottleBrakeImpl.parkingBrake(value=0/100)” and make it a “var” with modified “set” function (do what “ ThrottleBrakeImpl.parkingBrake(..)” function does) and a modified “get” function which reads the value from the PWM pin. Then server’s function “ ThrottleBrakeSystem.get\_parking\_brake\_state()” will call the custom get and will check for equality to 0 or 100 and return false/true accordingly.~~
  2. *The above cannot be applied because I set Int value and get back a String. The String SUCCESS which is returned it is used by the client to update the UI and at the moment it is designed this way.*

## Handbrake

### Client (DONE)

* 1. Handbrake applies to the rear wheels only. The user can activate it by clicking the icon. The handbrake must be active while the user touches the icon and inactive when the user releases the icon. The handbrake cannot stay active without touching it (like a physical handbrake with a button). For this, use the parking brake (will not be discussed here).
  2. The client will call server’s function “ThrottleBrakeSystem.set\_throttle\_brake\_system(id=incremental\_int, action=ACTION\_HANDBRAKE, value=0/100)”. The client will call server’s function in a non-blocking way. This means that the client will not wait for any answer by server. I suppose that during driving the handbrake is necessary (U-turns, etc) so it must not delay neither the server (car) nor the client (driver).
  3. The client will, also, call the server’s function “ThrottleBrakeSystem.get\_handbrake\_state()” to get the state of the handbrake in a blocking way. This function will not be used during driving for the reason I mentioned above but it can be used to reset the state of the icon during engine start/stop, etc.
  4. Do not forget to deactivate the parking brake if it is already enabled. According to my structure of “changeMotionInteractiveIconsStatus()” in client and “set\_throttle\_brake\_system(…) in server in combination with the way I call them and everything should be updated like a charm.

### Server (DONE)

* 1. Handbrake belongs to the ThrottleBrakeImpl class in hardware. So, the server already resets this kind of state for the car when the engine is stopped.
  2. Due to the handrake is applied from server’s function “ThrottleBrakeSystem.set\_throttle\_brake\_system(id=incremental\_int, action=ACTION\_HANDBRAKE, value=0/100)” the parking brake and anything else connected with the motion will be reset according to the “lastRequestId”’s command. I think that the return String of this function should not be used in a blocking way on client due to the delay. But this function should return this String (not be Unit) for debugging issues whenever I want.
  3. *At the moment the following could be based on server events and not from hardware events.*
  4. Server’s function “ThrottleBrakeSystem.get\_handbrake\_state” will call hardware’s “var ThrottleBrakeImpl.isHandbrakeActive”with modified “get” function. Server’s function will return a Boolean.

### Hardware

* 1. As said at server section, there will be a “ThrottleBrakeImpl.handbrake(value=0/100)” function which will apply the handbrake. The return String of this function should be the value of the pin, but if I cannot get this value it will return 0 or 100 according to the function argument or an error message according to the exception. Before returning the String value (not the exception message) and if and only if I could not get values from PWM pins, do not forget to “set” the appropriate value to the “var ThrottleBrakeImpl.isHandbrakeActive“.
  2. The “get” function of the “var ThrottleBrakeImpl.isHandbrakeActive“ will be modified also, because I think that I can get data back from PWM pins. If I cannot get data back from PWM pins then there will be a regular “get” function. This “var” will have a private “set” function.
  3. “ThrottleBrakeImpl” class should have a private function “applyPwmValues(motorXXPinYY, …) where XX is FR, FL, RR, RL and YY is 00, 01, 11, 12”. I know that the “enable” pin is missing from this setup but it will be added in during implementation. When this function is called using all arguments every time it will automatically unset the previous settings from the motors.

## Throttle/Brake/Neutral

### Client (DONE)

* 1. The “SeekBar” used for throttle/brake/neutral may need some redesign in its usage. The pwm value 0.50 means throttle for a previous value of 0.25 and means brake for a previous value of 0.75. Also, according to H-bridge the “pwm=0” means fast brake (brake pedal to the floor). So, 0 pwm value cannot be used for neutral.
  2. The new UX design I suggest (yeah, to myself!) is the following:
  + The default “SeekBar” value (when user lifts his finger) will be 5 (or 1, or a value that means no brake and no movement) instead of 50. The default value must have the car as neutral as possible.
  + The values that will be send to server are 20 (or the least value that can move the car), 40, 50, 65, 75, 80, 85, 90, 95, 100. The steps between these values must be tested in order to be sure that the car is moving as smooth as possible without having unexpected acceleration.
  + The “pwm=0” value are the fast brakes (brake pedal to the floor) and it must be selected by the user because brake-sliding the car makes the car uncontrollable. Also the brakes could be the negative or positive (the opposite of the direction the car is moving) value of the neutral (example: if car is moving forward and could stay neutral at value 10 the brake could be -10 and the opposite if car is moving backwards), but this needs a lot of testing using the final motors in which the car will run.

The requests should be done in a non-blocking way, so the client will not wait from a result from server functions.

When the user first touches the SeekBar it will call server's function “ThrottleBrakeSystem.set\_throttle\_brake\_system(id=incremental\_int, action=ACTION\_NEUTRAL, value=0)”. This way the car will be in freewheel mode. The client will update the motion related icons in a blocking way. This can be blocking because the user is not driving but he is getting prepared to do.

When the user lifts his finger from the SeekBar, the SeekBar itself will be set back to freewheeling value and call server's function “ThrottleBrakeSystem.set\_throttle\_brake\_system(id=incremental\_int, action=ACTION\_NEUTRAL, value=0)” in a non blocking way.

When the user moves the SeekBar's cursor back and forth the server's function “ThrottleBrakeSystem.set\_throttle\_brake\_system(id=incremental\_int, action=ACTION\_FORWARD/ACTION\_BACKWARD/ACTION\_NEUTRAL/ACTION\_BRAKING\_STILL, value=seekbar\_value)” will be called in a non-blocking way at specified values (see the beginning of this section).

### Server (DONE)

* 1. Every set job will be done using “ThrottleBrakeSystem.set\_throttle\_brake\_system(id=incremental\_int, action=ACTION\_FORWARD/ACTION\_BACKWARD/ACTION\_NEUTRAL/ACTION\_BRAKING\_STILL, value=0/seekbar\_value)”.
  2. *At the moment the following could be based on server events and not from hardware events.*
  3. Server’s function “ThrottleBrakeSystem.get\_motion\_state” will call hardware’s “var ThrottleBrakeImpl.isMovingForward and isMovingBackward and isStandingStill”with modified “get” function. Server’s function will return a String representation of the motion state. This state cannot be Boolean.

### Hardware

* 1. I have to find the appropriate H-bridge which supports brake stop and freewheeling stop. First, I suggest (to myself once again) to have some tests using the L298N I already have.

## Reverse

### Client

* 1. When the user clicks the reverse “ImageView” a “Boolean” variable will change state. This variable will change the “action” parameter at function “ThrottleBrakeSystem.set\_throttle\_brake\_system(id=incremental\_int, action=ACTION\_FORWARD/ACTION\_BACKWARD, value=seekbar\_value)”.
  2. Also, the “ImageView” will be activated or deactivated according to the state of a local variable. The server will know when car is moving backward and not when the car is going to move backward in the next throttle action. The default state for this “ImageView” will be false (means not backward).
  3. The reverse “ImageView” affects the motion but the “ImageView” itself is not affected by the motion UI items. So, this “ImageView” changes it’s state from function ““changeMotionInteractiveIconsStatus()”.

### Server

* 1. According to previous implementation designs this needs nothing more to work.

### Hardware

* 1. TODO.

## Cruise Control

### Explanation

* 1. This will allow car to travel in a specific speed until some events will occur.
  2. The user should have his throttle-finger on the throttle seekbar otherwise the neutral state will be applied. When the user long-clicks the cruise control “ImageView” to activate it the throttle seekbar will not return to neutral state.
  3. The cruise control will be deactivated only if the user touches the throttle seekbar. If the cruise control is activated and the user long-clicks the “ImageView” a Toast will say “Cruise Control activated.\nUse throttle to deactivate it”.
  4. When the user just clicks the cruise control “ImageView” a Toast will inform the user about the long-click action.
  5. The cruise control activation/deactivation just happens at the client side and the server does not give an F-word, so the server will not be informed.

### Client (DONE)

* 1. At “RCControllerActivity” there will be a “var cruiseControlState: Boolean”. This will be in the main “Activity” because it has nothing to do with the network. This value will be set to “true” when the user long-clicks the cruise control “ImageView”. If the “cruiseControlState” is already “true” a Toast will show info (see “Explanation” section above) .It will set to “false” when the user starts to change the throttle value through the “onStartTrackingTouch(...)” function.
  2. At throttle “SeekBar”, when “onStopTrackingTouch(...)” is executed it will be an if-else clause to determine if the “SeekBar”’s value should return to neutral or stay where it is.

### Server (DONE)

* 1. Nothing to do here.

### Hardware (DONE)

Nothing to do here.

## Steering Left/Right/Neutral

### Client (DONE)

* 1. The SeekBar’s cursor default and neutral position is in the middle. When in the middle the car's wheels are looking straight.
  2. Here, the communication between the client and the server should be executed in a non-blocking way.
  3. The client will call server's function «SteeringSystem.set\_direction(id=incremental\_int, action=ACTION\_RIGHT/ACTION\_LEFT/ACTION\_STRAIGHT, value=0/seekbar\_value)”.
  4. Like in throttle/brake control I will have a different incremental var for id. This var «lastSteeringRequestId» will be sent as an id to the previous server function.
  5. Do not forget to reset the var to 0 just like in throttle/brake and also do not forget to rename the throttle/brake id to «lastMotionRequestId». The renaming should be done in client only because at server I call the «lastRequestId» using the class first, so it is difficult to get confused.

### Server (DONE)

* 1. Server will have a function «SteeringSystem.set\_direction(id=incremental\_int, action=ACTION\_RIGHT/ACTION\_LEFT/ACTION\_STRAIGHT, value=0/seekbar\_value)” which get data from the client and apply it the hardware.
  2. Here I use a var «lastRequestId» like in «ThrottleSystem» to make sure the last request from the client is applied. Also, this var, just like «ThrlottleSystem»’s must be reset when the engine is off.

### Hardware

# Electric Car Functions

## Lights Front (Low / Normal / High) and Back

### Client

### Server

### Hardware

## Brake Lights (Back)

### Client

### Server

### Hardware

## Turn Lights (Left / Right / Straight)

### Client

The client will have a “ var ClientTriggeredRequests.directionLight=left/right/straight)” with modified “get” and “set” functions. The modified “set” function will call server’s function “ElectricsSystem.set\_direction\_light(direction=left/right/straight) and the modified “get” function will call server’s function “ElectricsSystem.get\_direction\_light()”. Every call from the modified “set” and “get” will be executed in a blocking way.

When the user long-clicks a left/right direction light, the app will:

* get the value from “ClientTriggeredRequests.directionLight” and:
  + if returns left/right, then it set the value “straight” to the “ClientTriggeredRequests.directionLight” in order to turn it off
  + if returns right/left, then it set the value “left/right” to the “ClientTriggeredRequests.directionLight” in order to turn it on
* get the value from “ClientTriggeredRequests.directionLight” once again and update the UI turn light items according to the returned value.

*TODO: At the moment the turn light “ImageView” stays on and it is not blinking. In order to make it blink I need to start a Timer and stop it when the led must be off.*

### Server

* 1. The server will be listening “ElectricsSystem.set\_direction\_light(direction=left/right/straight)” for events. This is how this function works:
* if direction is “left”/“right” then call “ElectricsImpl.handleLeds(left/right=1, right/left=0)”
* if direction is “straight” then call “ElectricsImpl.handleLeds(left=0, right=0)”
  1. Also it will have a function “ElectricsSystem.get\_direction\_light()” which returns the turn light direction. This function works as following:
* if “ElectricsImpl.lights[“left”/“right”]=1” and “Electrics.lights[“right”/“left”]=0 returns “left”/“right”
  + else if “ElectricsImpl.lights[“left”]=0” and “Electrics.lights[“right”]=0” returns “straight”
  + else returns “error”

### Hardware

The “ElectricsImpl” singleton will have a “private val \_lights:MutableMap<String,Boolean>” to control the LEDs and a “val lights:Map =\_lights” just to read the LEDs’ state whenever someone wants to.

Also, it will have a “reset()” function to set all LEDs’ values to “false” as a reset state.

Finally, the basic function of this class is the “synchronized handleLeds(…). This function is written with “...” as argument because I don’t know how many argument it will have. The arguments will have default values “-1”. The arguments represent the value for each LED. The main job of this function is to update the “\_lights” “MutableMap” and apply all the values through the Shift Register IC. The “\_lights” will be update according to the rule:

* if argument is “1” or “0” update the “\_lights” value
* else if the argument is “-1” leave the “\_lights” value as already is

I have to test if Shift Register IC accepts pwm values, else I have to modify the circuit to the LED with resistors in order to achieve the brightness grading. (*Note to myself: Should I use any MCPXXXXX IC instead of Shift Register?*)

*TODO: At the moment the turn light stays on and it is not blinking. In order to make it blink I need to start a Timer/TimerTask and stop it when the led must be off.*

To remember how the shift register works see my example at Tinkercad.

## Emergency Lights

### Client

### Server

### Hardware