Smart

Device

Dashboard

Detail Design

### PREPARED BY

Jeevan Acharya

Web Engineering SP22S CPSC-458-01

# 1. Project Overview

The Smart Device Dashboard is a web-based React.js portal that allows a user to see and control all their smart home devices made by the TP-Link company. Today the only way to check and control these devices in your home is based on a mobile application called Kasa, or via Amazon Alexa or Google Home, there is no option for a desktop / web based control.

This poses some challenges which this project hopes to overcome. The Google Home or Amazon Alexa applications have value but they are disruptive, meaning you have to voice out the commands and often the device repeats the status of your command. If you are in a meeting or a conference call this is not desirable as it interrupts others. The mobile application is great when you are on the move or away from home or office, however if you are sitting at your desk you have the inconvenience of pulling out your phone, finding the Kasa application, and then controlling the device. This process is repeated if you want to check or control the device again which can become extremely annoying, not just for the user but also for others that witness your constant checking on the phone.

During meetings and conference calls it is considered disrespectful for someone to actively check their phone. On the other hand people are used to seeing people having their laptops open and since they cannot see your screen, would not feel disrespected if a user is checking the website to control their devices.

A side benefit of having a website is the ability of other devices that have internet connectivity participating. Kasa application for mobile devices has many benefits in its rich user interface although it also forces the developers of the application to restrict it to certain devices such as Android devices or Apple devices that are running specific versions of the OS. If the Apple device you have is not on the latest iOS or if the Android application cannot be downloaded to your specific device (e.g. Android Auto Car Stereo systems) you are out of luck. Utilising our Smart Device Dashboard we eliminate many of these restrictions and allow any device that has a modern browser to participate. The range of devices increases from many mobile devices, chromebooks, modern aftermarket car stereos, most smart tvs, and laptops or desktops. The segment of users increased significantly thanks to the humble web browser.

# 2. Obstacles

TP-Link does not provide an official API that developers can use to integrate their applications with the devices. There is however an unofficial API called OPENTPL by stickyMonkeyFlowerLabs available as a freemium on RapidAPI website. This does have restrictions in that it mentioned that only HS-110, HS-100, and KP-115 Smart Plugs and the HS-200 Smart Switch have been tested. There is a risk that these APIs may not work fully or have some issues, until tested no work around has been planned.

Another obstacle is having to purchase these devices for testing. My parents home has a few of these switches that can be used, and just in case I have purchased one smart plug device for demonstration.

In person demos have restrictions because Smart Switches need to be wired in and would therefore be unavailable to show live, and the smart plugs will need to connect to a local wifi with internet connectivity. There are a few options to overcome this challenge:

1. Utilise my parents smart home to perform controls and display the Kasa Application on a phone to see the status from the Dashboard and the Phone Application,
2. A Smart Plug could be brought in connected to a lamp and utilising my phone's hotspot.

## Kasa Account

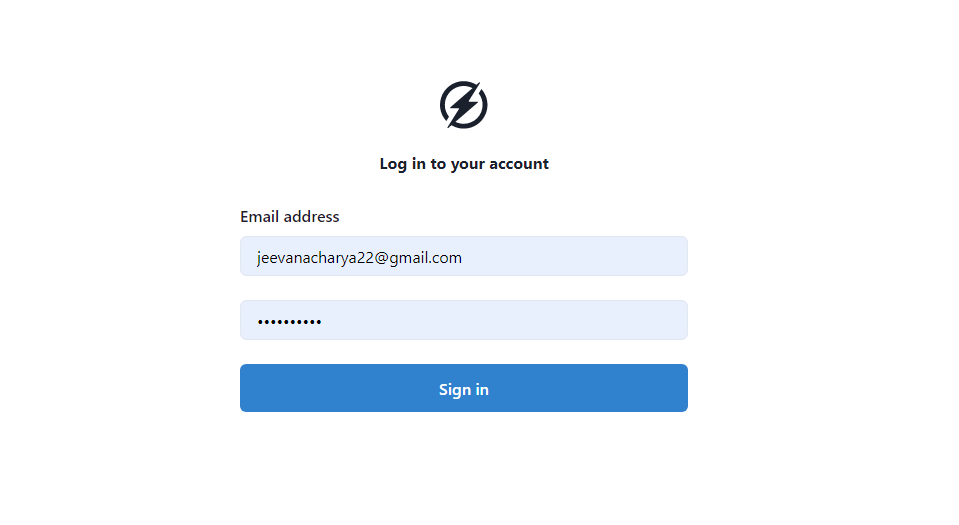
The OPENTL assumes that an account via Kasa application has been created and all devices for that account added. It is critical that this step has been done prior to using the dashboard.

The OPENTL does not allow for a user to add a smart device, only controlling the device or monitoring its status.

# 4. Smart Device Dashboard

This section will present example screens to highlight the use case and overall layout on both a desktop portal and a mobile browser. The final design and layout may change based on several factors such the icon availability, user experience, and time to complete the project. What is presented here is the ideal scenario and certain functions may be implemented in a later stage.

## Initial Screen



For security reasons no user id and/or password is stored. The OPENTL api will need to provide these fields prior to executing any further commands. For this reason the first screen that is displayed is the Login screen with 2 fields for the user id and password. These values must match the Kasa account of the user as they will be passed in the OPENTL API[[1]](#footnote-0) command create\_user.

The API POSTs to the URL with a JSON body of the form:

const request = require('request');

const options = {

method: 'POST',

url: 'https://opentpl1.p.rapidapi.com/dev/tplapi',

headers: {

'content-type': 'application/json',

'X-RapidAPI-Host': 'opentpl1.p.rapidapi.com',

'X-RapidAPI-Key': '<RapidAPI Key>',

useQueryString: true

},

body: {

body: {command: 'create\_account', password: '<kasa password>'},

uuid: '<kasa id>'

},

json: true

};

request(options, function (error, response, body) {

if (error) throw new Error(error);

console.log(body);

});

**<RapidAPI Key>** = the unique provided to your RapidAPI account

**<kasa password>** = the password you have for your kasa account

**<kasa id>** = the id you use to log in to your kasa account

Although the command says create\_user, a new user is not actually created in kasa, it is more to establish a “session” or connection to TP\_LINK. This command only needs to be entered once after the user logins in to the Dashboard. Once done the delete\_user command is issued to disconnect the user from the session during logout.

Similarly a delete\_user will not actually delete the user account from the Kasa application.

## Smart Device Status



Once the user has logged in a list of all smart devices shall be displayed. In addition, a toggle icon is displayed to show if the device is on or off, and allow the user to switch state when pressed. The name of the device is the name retrieved from the OPENTPL response. These names are set up by the user in their Kasa application.

The device Info is not currently returned in the response from OPENTPL, however after contacting the support line this field may be provided when coding starts. If this is missing at the time coding starts this will be blank.

UPDATE: An enhancement to the OPENTPL API was provided by the developer to return the device type, however in order to get the device type we need to call the command plug\_names followed by the status command.

The calls to OPENTPL have a slow response and when calling the commands twice it is noticeable on the screen, I have contacted the support to see if there is anything that can be done.

### Filter Options

On the left hand there are links which will ensure that the list displayed can be filtered based on the type of device.

### 

The filters are based on:

* Smart Devices
  + Display all devices
* Smart Light
  + Display only LB100 devices[[2]](#footnote-1)
* Smart Switch
  + Display only HS200 devices
* Smart Plugs
  + Assumes any devices not listed above are smart plugs.
  + The only ones know at this time are HS110, HS100, HS300[[3]](#footnote-2), and KP115

### Future Enhancement

Depending on the time allowed it would be nice to replace the name of the device (HS200) with an icon for that device. This improves the look and feel of the dashboard but it will increase the maintenance of the dashboard in the long run. If a new device type is returned in the response then the image for that device would not be found.

# OPENTPL REST API

The OPENTPL RestAPI can be found here: <https://rapidapi.com/clenox/api/opentpl1/details>

In order to use this API an account needs to be created on the RapidAPI website. Once the account is created a unique RapidAPI key is provided to the account that needs to be used for each API request the developer may use.

When using the OpenTPL API in the code, it first needs to authenticate with the RapidAPI website using the unique key associated with the account. The OPENTPL API subscription allows for 300 API requests in a month for free before a charge to a credit card is placed.

The interaction is controlled by the RapidAPI Key that is used when calling any of the OPENTPL APIs.

The basic flow for using the OPENTPL is:

* Establish a connection with the RapidAPI and TP-Link website with your account using the unique RapidAPI key. Use the command: create\_user
* Get the names of devices associated with the Kasa account. Use command

const request = require('request');

const options = {

method: 'POST',

url: 'https://opentpl1.p.rapidapi.com/dev/tplapi',

headers: {

'content-type': 'application/json',

'X-RapidAPI-Host': 'opentpl1.p.rapidapi.com',

'X-RapidAPI-Key': '<RapidAPI Key>',

useQueryString: true

},

body: {body: {command: 'plug\_names'}, uuid: '<kasa id>'},

json: true

};

request(options, function (error, response, body) {

if (error) throw new Error(error);

console.log(body);

});

The response to this command is a list of devices with their unique id and other details:

**{**

"data":**{**

"90061A382A63189289DAECEC0E6EE37F194D0508":"Jeevans Light"

"90061AA9D9F6E5A2E6C1A7274736E93517C7FFF8":"Front Lights"

"900648EA0F8D71766ED3E8C4CD3928771D911478":"TP-LINK\_Power Strip\_00EC"

"90066021B1362AAD17D6FA3E8D9B3DFC17BF4818":"Master Bedroom"

**}**

"Error":false

**}**

* For each of the devices the status of the device can be obtained by issuing a status command. An example to get the status of Jeevans Light is provided below

{

"body": {

"command": "status",

"plugid": "90061A382A63189289DAECEC0E6EE37F194D0508"

},

"uuid": "<kasa\_id>"

}

The response to this command is:

**{**

"error":false

"state":"OFF"

"status":"online"

**}**

The state indicates if the light is on or off, the status indicates if the device is on line, meaning it is connected to the internet,

* The device can be turned on or off by issuing the ON or OFF command,:

{

"body": {

"command": "ON",

"plugid": "90061A382A63189289DAECEC0E6EE37F194D0508"

},

"uuid": "<kasa\_id>"

}

The response is:

**{**

"error":false

"status":"ON"

**}**

* **Once done the user should log out using delete\_user**

**{**

**"body": {**

**"command": "del\_account"**

**},**

**"uuid": "**<kasa\_id>**"**

**}**

# React

I am utilising the chakra-ui framework which provides the icons for the toggle switches and boxes which I use for displaying the devices. In the box I can insert the icon to represent plugs, bulbs, and switches. The icons that can be selected are shown here: [React Icons (react-icons.github.io)](https://react-icons.github.io/react-icons/icons?name=fa)

## Components

* Common
  + Device - Information about the individual device
  + Device Card - The list of all devices
* LoginPage
* SmartDeviceList
* ShowAllDevices
* ShowSmartPlug
* ShowLightSwitch

## How Local Storage Is Used

* I do not utilise any local storage.

Regarding the RapidAPI key, since placing this key in the local storage exposes a risk that anyone with access to Git can see my key and therefore reuse it it should not be placed here. There are two options for resolving this:

1. Use the .env file to place my RapiAPI key. This is the method I have chosen, however this has the risk that anyone that opens the console log can see in real time my key and therefore open a risk of misuse. Due to timeline of getting a working project I will use this method since this .env wont be in git, and also after the demo I can regenerate the key making the old one useless.
2. Designing a backend database. Here I would have to create a MySQL database and have a table that could hold the key. My code would then have to log in to the MySQL DB to retrieve this key, either using a REST API (Ideal) or a connection directly. This would be the ideal case as I can hide everything but requires more coding than time permits.

## Important Things to Do

Use the useEffect function to refresh our page with updates. Additionally, I could get around this by using the NextJS features.

1. Details found in the following link: https://rapidapi.com/clenox/api/opentpl1/details [↑](#footnote-ref-0)
2. Since I do not own any smart light bulbs this will be coded but unverified [↑](#footnote-ref-1)
3. The OPENTPL states that this device type has not been verified to work. [↑](#footnote-ref-2)