# Step 1: Develop new C-Panel with 3 new GPS reports

-API with the GPSWOX server

<https://gpswox.api-docs.io/v1.0/welcome/introduction>

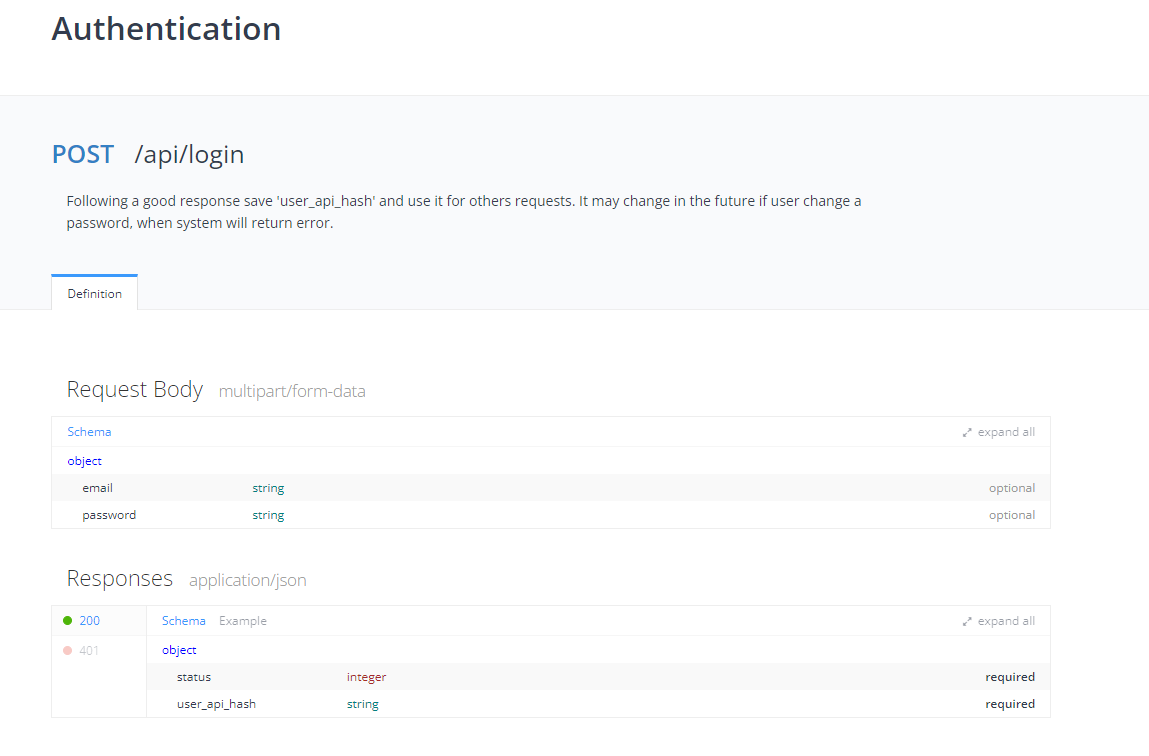
## Login function to New C-Panel with previous credential

### Requirement Overview

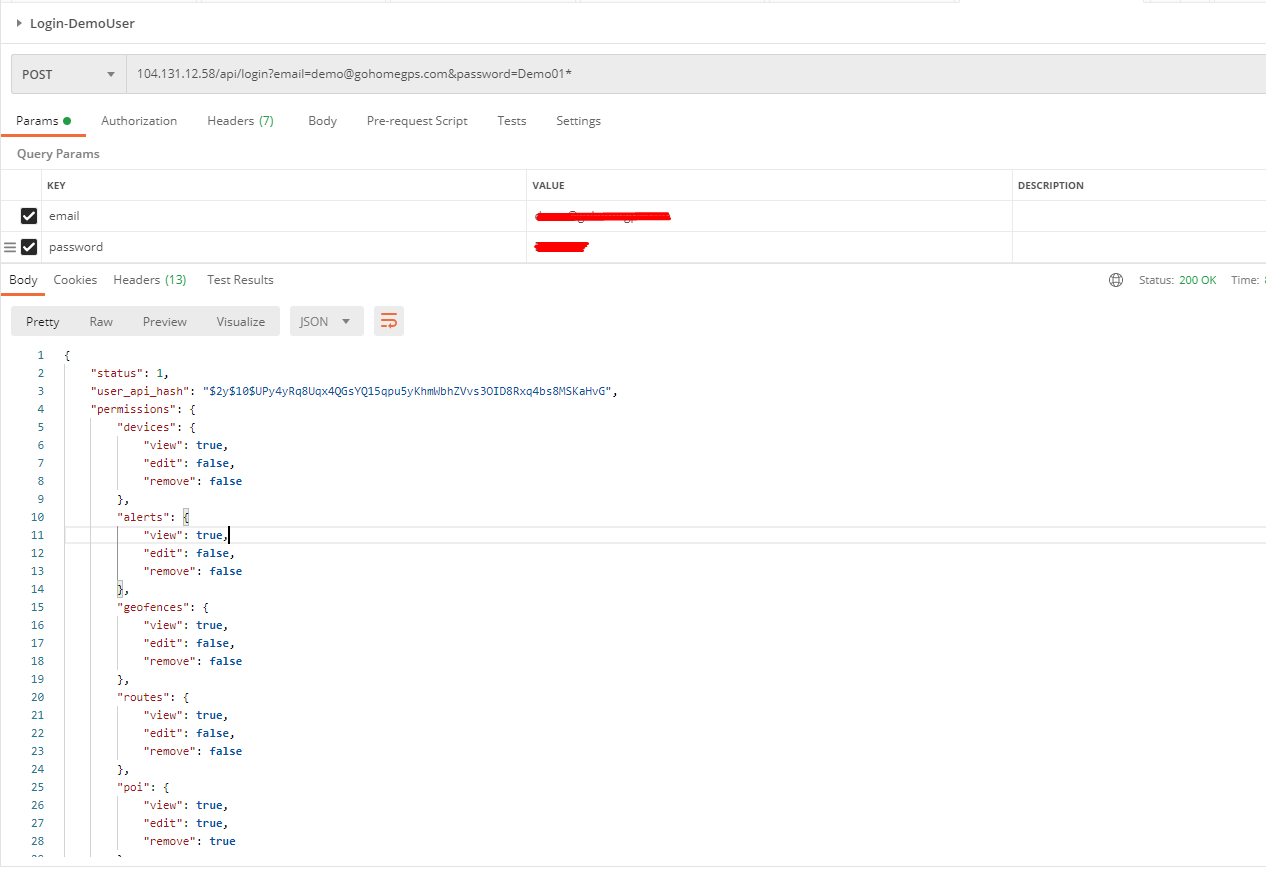
Customer already have credential on previous c-panel and new c-panel have to allow user to login to new c-panel with previous credential

### API Research

<https://gpswox.api-docs.io/v1.0/authentication/tWmFcYTX3D6BpcR32>



**API Response**

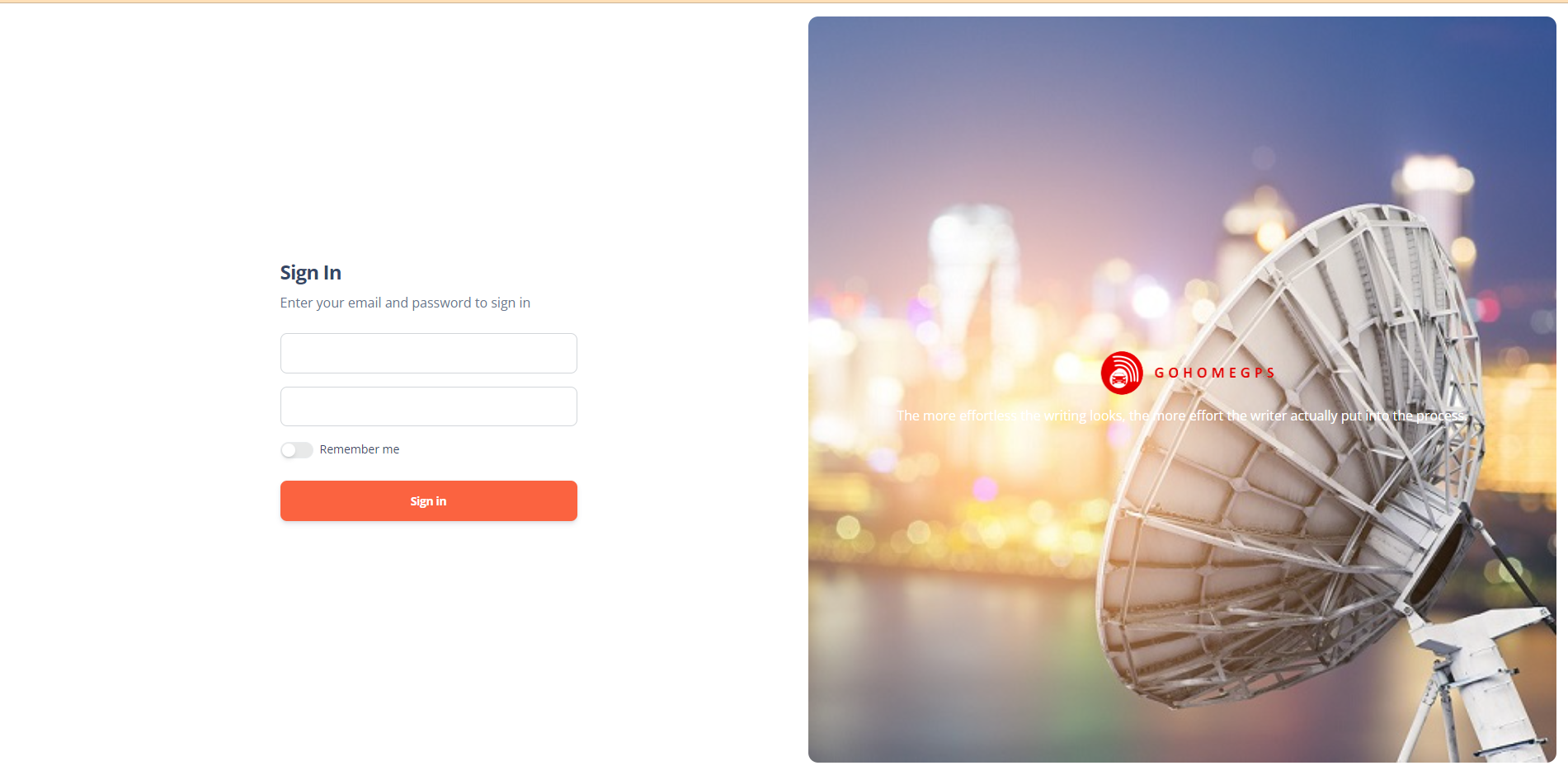


API return user\_api\_value and each permissions.

We have to user\_api\_value on other API s.

### Implement

#### UI building



#### Code Implement

* Route

    Route::post('/login', [LoginController::class, 'login'])->middleware('guest')->name('login.perform');

* Controller

public function login(Request $request)

    {

        $credentials = $request->validate([

            'email' => ['required', 'email'],

            'password' => ['required'],

        ]);

        try{

            $client = new Client(['timeout' => 180]);

            $response = $client->post('http://104.131.12.58/api/login', [

                'headers' => [

                    'Content-Type' => 'application/json',

                ],

                'json' => [

                    'email' => $request->input('email'),

                    'password' => $request->input('password'),

                ],

            ]);

            $responseData = json\_decode($response->getBody()->getContents(), true);

            if($responseData["status"]==1 && $responseData["permissions"]["reports"]["view"] == true ){

                Auth::attempt(['email' => "admin@argon.com", 'password' => "secret"]);

                session(['user\_api\_hash' => $responseData["user\_api\_hash"]]);

                return redirect()->intended('usermanagement');

            }

            else{

                return back()->withErrors([

                    'permission' => "You don't have permission to access to this service",

                ]);

            }

        }

        catch(ClientException $e)

        {

            if ($e->getResponse()->getStatusCode() === 401) {

                // handle the 401 Unauthorized response here

                return back()->withErrors([

                    'email' => 'The provided credentials do not match our records.',

                ]);

            } else {

                // handle other 4xx errors here

                return view('error')->with('message', 'An error occurred while processing your request');

            }

        }

}

Code check that user have status value 1 and have permission to view reports

If user insert wrong email or password, the page show alert about that.

Also if user have no permission to review report, the page show alert about that too.

When valid user try login, c-panel save user’s user\_api\_hash value for data analyze.

* View

<form role="form" method="POST" action="{{ route('login.perform') }}">

                                        @csrf

                                        @method('post')

                                        @error('permission') <p class="text-danger text-xs pt-1"> {{$message}} </p>@enderror

                                        <div class="flex flex-col mb-3">

                                            <input type="email" name="email" class="form-control form-control-lg" value="{{ old('email') ?? 'admin@argon.com' }}" aria-label="Email">

                                            @error('email') <p class="text-danger text-xs pt-1"> {{$message}} </p>@enderror

                                        </div>

                                        <div class="flex flex-col mb-3">

                                            <input type="password" name="password" class="form-control form-control-lg" aria-label="Password" value="secret" >

                                            @error('password') <p class="text-danger text-xs pt-1"> {{$message}} </p>@enderror

                                        </div>

                                        <div class="form-check form-switch">

                                            <input class="form-check-input" name="remember" type="checkbox" id="rememberMe">

                                            <label class="form-check-label" for="rememberMe">Remember me</label>

                                        </div>

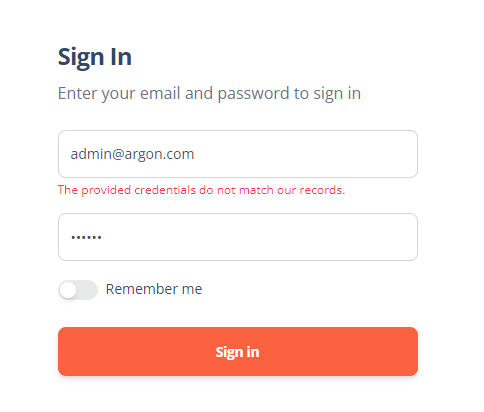
                                        <div class="text-center">

                                            <button type="submit" class="btn btn-lg btn-primary btn-lg w-100 mt-4 mb-0">Sign in</button>

                                        </div>

                                    </form>

### Test



## New 3 report pages

Taking information from the GPSWOX server database and analyze.

### Green Driving Report

#### Requirement Research

This Report collects the negative driving score of the drivers and their kilometers traveled, to carry out a calculation that allows assigning a score to said driving.

The negative points are: High speed, High speed time, Aggressive Braking, Aggressive Cornering, Aggressive Acceleration.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Driver Name | Unit Name | kilometers traveled | speeding count | speeding duration | aggressive acceleration | aggressive braking | aggressive cornering | driver score |
|  |  |  |  |  |  |  |  |  |

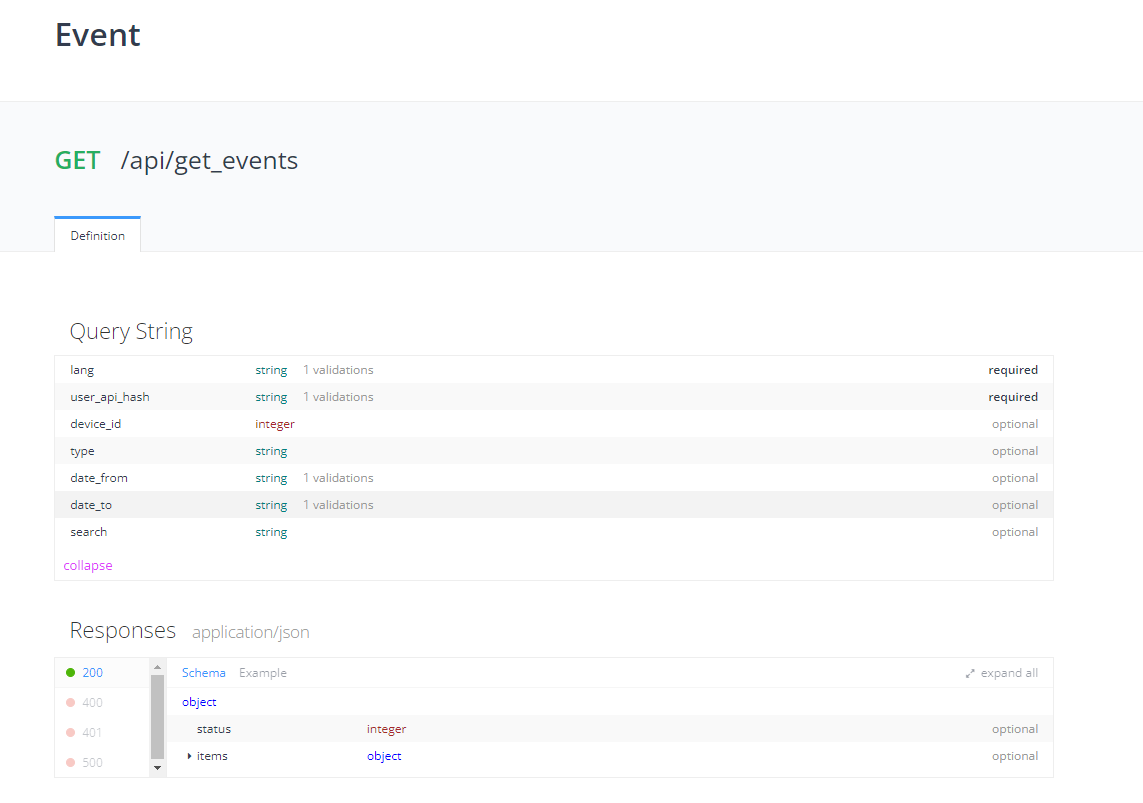
#### API Research

When user set event on previous c-panel, user set message for event.

For example, about Aggressive event, set message as “maxacceleration”.

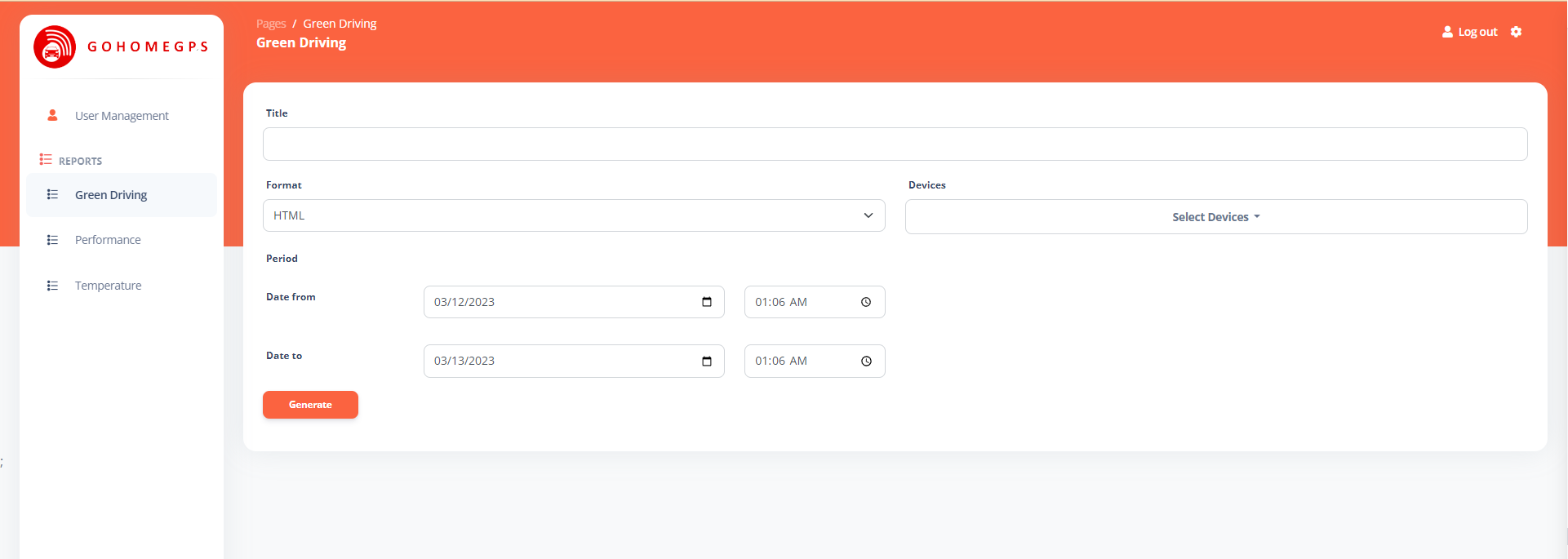
By check this message, we can check event is occurred or not.

<https://gpswox.api-docs.io/v1.0/event/zKmb3BxcJFjprLTXp>



Events API return many pages, so to get all events, we have to get all info from each pages

#### UI Building



#### Implement

* Route

\routes\web.php

    Route::get('/{page}', [PageController::class, 'index'])->name('page');

* Controller

\app\Http\Controllers\PageController.php

public function greendriving()

    {

        $response = Http::get("104.131.12.58/api/get\_devices?user\_api\_hash=".session('user\_api\_hash'));

        $jsonData = $response->json();

        $devices = $jsonData[0]['items'];

        return view("pages.green-driving", compact('devices'));

    }

Get user\_api\_hash from session and get all vehicles data and send to view.

* View

\resources\views\pages\green-driving.blade.php

* Data Analyze

\*All data is analyzed in app.blade.php

\resources\views\layouts\app.blade.php

for (let i = 0; i < devicesData.length; i++) {

                        const dData = JSON.parse(devicesData[i]);

                        let totalDistance = 0;

                        let totalMaxAcceleration = 0;

                        let totalMaxBraking = 0;

                        let totalMaxCornering = 0;

                        let totalHighSpeed = 0;

                        const historyApiURL = `http://104.131.12.58/api/get\_history?user\_api\_hash=${user\_api\_hash\_value}&device\_id=${dData.deviceID}&from\_date=${fromDate}&from\_time=${fromTime}&to\_date=${toDate}&to\_time=${toTime}`;

                        const historyResponse = await fetch(historyApiURL);

                        const historyData = await historyResponse.json();

                        totalDistance = historyData.distance\_sum;

                        const apiURL = `http://104.131.12.58/api/get\_events?user\_api\_hash=${user\_api\_hash\_value}&device\_id=${dData.deviceID}&from\_date=${fromDate}&to\_date=${toDate}`;

                        const response = await fetch(apiURL);

                        const data = await response.json();

                        const totalPage = data.items.last\_page;

                        for (let j = 1; j <= totalPage; j++) {

                            const EventApiURL = `http://104.131.12.58/api/get\_events?page=${j}&user\_api\_hash=${user\_api\_hash\_value}&device\_id=${dData.deviceID}&date\_from=${fromDate}&date\_to=${toDate}`;

                            const Eventresponse = await fetch(EventApiURL);

                            const Eventdata = await Eventresponse.json();

                            const Events = Eventdata.items.data;

                            if (Array.isArray(Events)) {

                                Events.forEach((event) => {

                                    if (event["message"].replace(/\s/g, '').toLowerCase() == "maxacceleration")

                                        totalMaxAcceleration++;

                                    if (event["message"].replace(/\s/g, '').toLowerCase() == "maxbraking")

                                        totalMaxBraking++;

                                    if (event["message"].replace(/\s/g, '').toLowerCase() == "maxcornering")

                                        totalMaxCornering++;

                                    if (event["message"].replace(/\s/g, '').toLowerCase() == "highspeed")

                                        totalHighSpeed++;

                                });

                            }

                        }

                        const eachDeviceData = {

                            deviceID: dData.deviceID,

                            driverName: dData.driverName,

                            deviceName: dData.deviceName,

                            totalDistance: totalDistance,

                            totalHighSpeed: totalHighSpeed,

                            totalMaxAcceleration: totalMaxAcceleration,

                            totalMaxBraking: totalMaxBraking,

                            totalMaxCornering: totalMaxCornering,

                        };

                        totalOutputData.push(eachDeviceData);

                    }

#### Result

* HTML

if(outputFormat == "html")

                    {

                        let tbodyOutputData = "";

                        for (let k = 0; k<totalOutputData.length; k++){

                            console.log(totalOutputData[k]);

                            tbodyOutputData += "<tr>";

                            tbodyOutputData += `<td>${totalOutputData[k].driverName}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].deviceName}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].totalDistance}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].totalHighSpeed}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].totalMaxAcceleration}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].totalMaxBraking}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].totalMaxCornering}</td>`;

                            tbodyOutputData += "</tr>";

                        }

                        let tableOutput = `

                        <table class="table table-bordered">

                            <thead>

                                <tr>

                                    <th>Driver Name</th>

                                    <th>Device Name</th>

                                    <th>Total Distance</th>

                                    <th>Hign Speed</th>

                                    <th>Max Acceleration</th>

                                    <th>Max Braking</th>

                                    <th>Max Cornering</th>

                                </tr>

                            </thead>

                            <tbody>

                            ${tbodyOutputData}

                            </tbody>

                        </table>

                        `

                        GenerateHTML(title, fromDate, fromTime, toDate, toTime, tableOutput);

                    }

* XML

if(outputFormat == "xml"){

                        let xmlOutput = "<root>";

                        for (let k = 0; k<totalOutputData.length; k++){

                            console.log(totalOutputData[k]);

                            xmlOutput += "<record>";

                            xmlOutput += `<driverName>${totalOutputData[k].driverName}</driverName>`;

                            xmlOutput += `<deviceName>${totalOutputData[k].deviceName}</deviceName>`;

                            xmlOutput += `<totalDistance>${totalOutputData[k].totalDistance}</totalDistance>`;

                            xmlOutput += `<totalHighSpeed>${totalOutputData[k].totalHighSpeed}</totalHighSpeed>`;

                            xmlOutput += `<totalMaxAcceleration>${totalOutputData[k].totalMaxAcceleration}</totalMaxAcceleration>`;

                            xmlOutput += `<totalMaxBraking>${totalOutputData[k].totalMaxBraking}</totalMaxBraking>`;

                            xmlOutput += `<totalMaxCornering>${totalOutputData[k].totalMaxCornering}</totalMaxCornering>`;

                            xmlOutput += "</record>";

                        }

                        xmlOutput += "</root>";

                        GenerateXML(title, fromDate, fromTime, toDate, toTime, xmlOutput)

                    }

* PDF

if(outputFormat == "pdf"){

                        let tbodyOutputData = "";

                        for (let k = 0; k<totalOutputData.length; k++){

                            console.log(totalOutputData[k]);

                            tbodyOutputData += "<tr>";

                            tbodyOutputData += `<td>${totalOutputData[k].driverName}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].deviceName}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].totalDistance}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].totalHighSpeed}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].totalMaxAcceleration}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].totalMaxBraking}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].totalMaxCornering}</td>`;

                            tbodyOutputData += "</tr>";

                        }

                        let tableOutput = `

                        <div class="container">

                        <table class="ptable table-bordered">

                            <thead>

                                <tr>

                                    <th>Driver Name</th>

                                    <th>Device Name</th>

                                    <th>Total Distance</th>

                                    <th>High Speed</th>

                                    <th>Max Acceleration</th>

                                    <th>Max Braking</th>

                                    <th>Max Cornering</th>

                                </tr>

                            </thead>

                            <tbody>

                            ${tbodyOutputData}

                            </tbody>

                        </table>

                        </div>

                        `

                        GeneratePDF(title, fromDate, fromTime, toDate, toTime, tableOutput)

                    }

* CSV

if(outputFormat == "csv"){

                        GenerateCSV(title, fromDate, fromTime, toDate, toTime, totalOutputData)

                    }

### Performance Report

#### Requirement Research

This report takes into account the driver, units, kilometers, gallon consumption, unit performance, kilometer cost and total route cost.

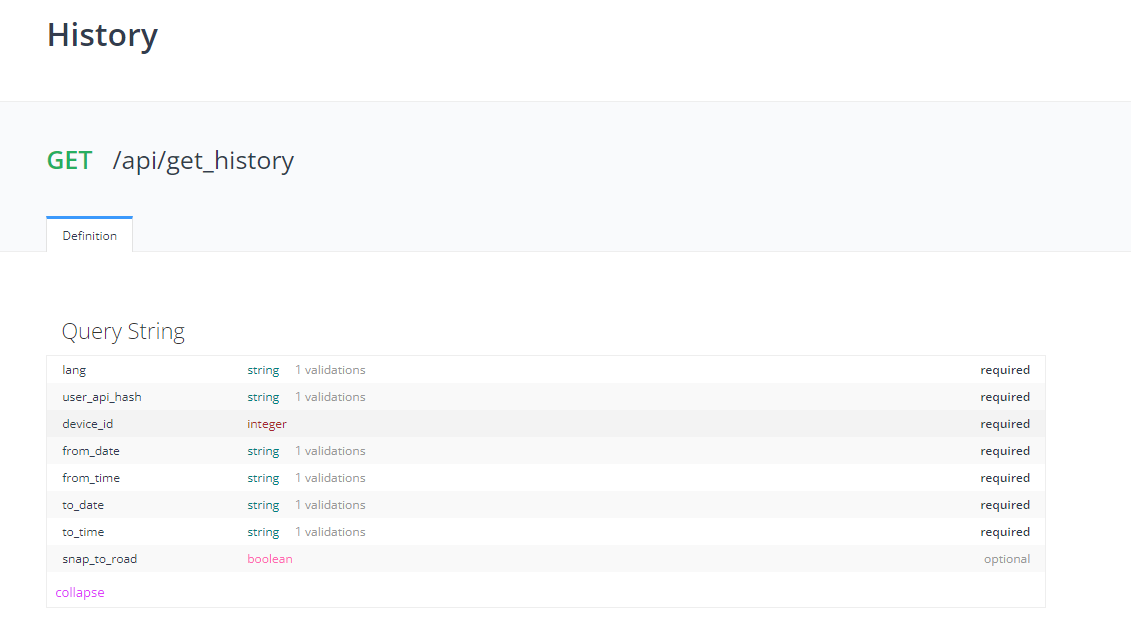
-To calculate cost, we have to add part to set Gallon price

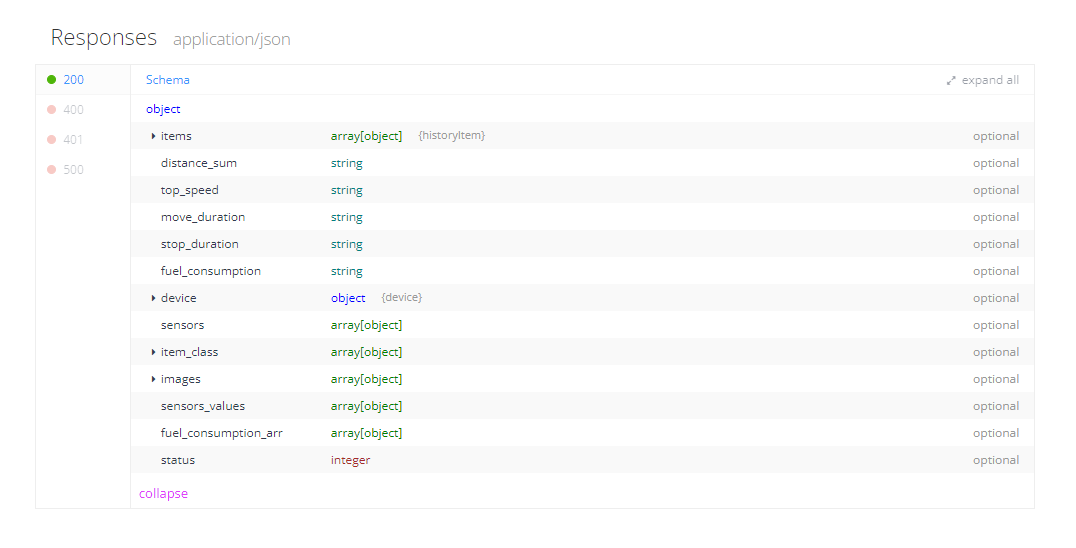
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Driver Name | Unit name | Kilometers traveled | Gallon Consumption | Unit Performance | Kilometer Cost | Total route Cost |
|  |  | Km | Gallon  (Value come from GPS as Liter  Have to change this to Gallon unit  1gallon = 3.785411784 liter) | Gallon Consumption / Kilometers traveled | Gallon Consumption \* Gallon Price / Kilocoeters traveled | Gallon Consumption \* Gallon Price |

#### API Research

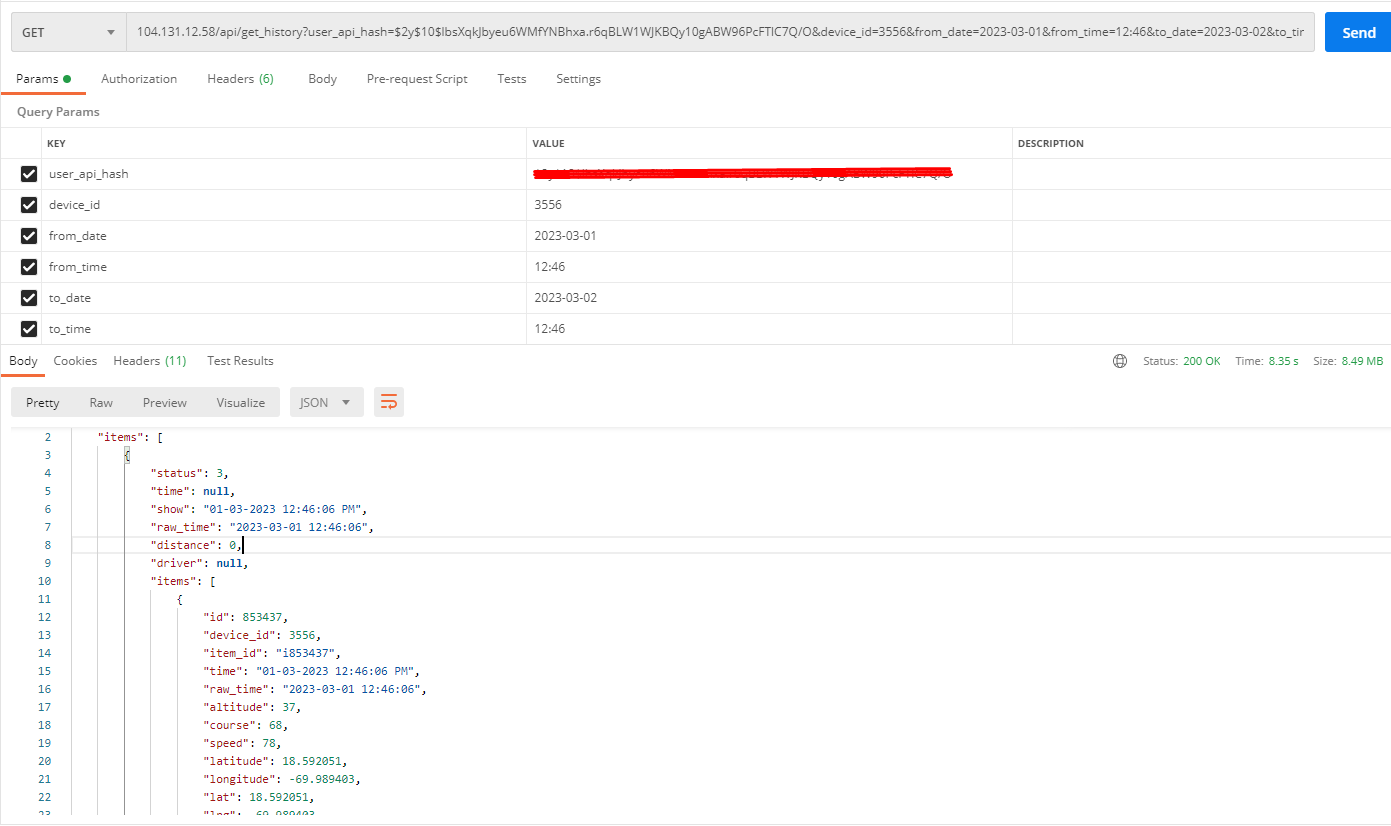
To get total traveled distance, we have to analyze vehicle’s history

<https://gpswox.api-docs.io/v1.0/history/CpN89SKgZDa27nhsi>

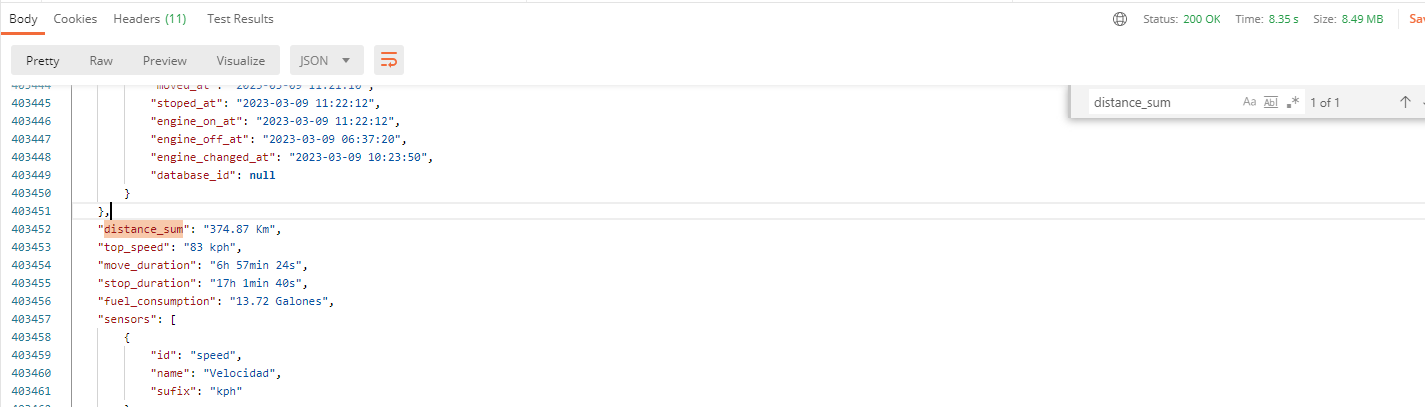




By provide user\_api\_hash, device\_id and period, we can get history of each device.

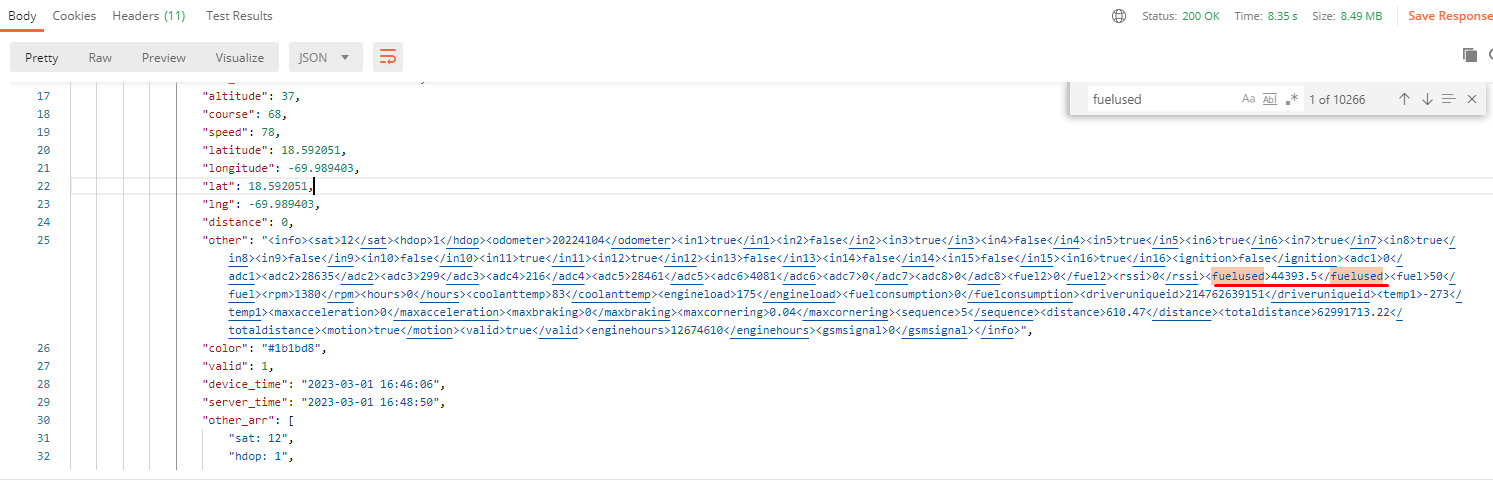


Total distance can be get like this

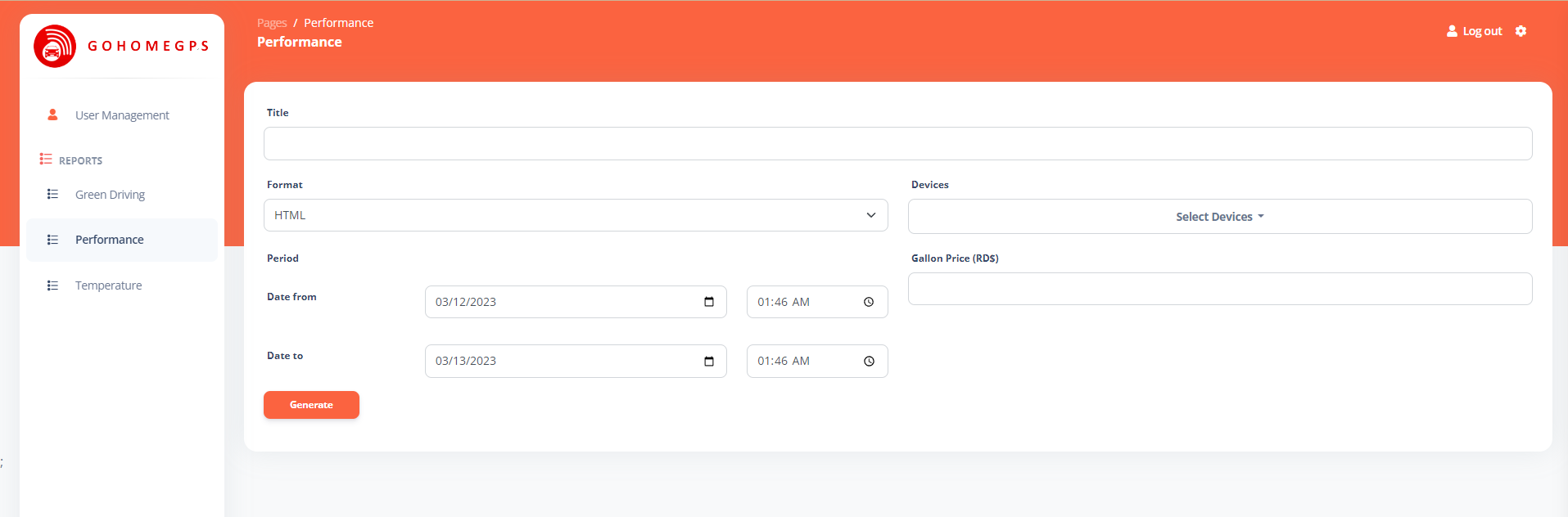


The amount of fuel used can be calculated by init fuel record - last fuel record

Fuel record is inclueded in API response like below



#### UI Building



You can set title, report result format, period and gallon price.

Also can select devices that you want to generate report.

#### Implement

* Route

\routes\web.php

    Route::get('/{page}', [PageController::class, 'index'])->name('page');

* Controller

\app\Http\Controllers\PageController.php

public function performance()

    {

        $response = Http::get('104.131.12.58/api/get\_devices?user\_api\_hash='.session('user\_api\_hash'));

        $jsonData = $response->json();

        $devices = $jsonData[0]['items'];

        return view("pages.performance", compact('devices'));

    }

* View

\resources\views\pages\performance.blade.php

* Data Analyze

for (let i = 0; i < devicesData.length; i++) {

                        const dData = JSON.parse(devicesData[i]);

                        let totalDistance = 0;

                        let totalFuelUsed = 0;

                        let unitPerformance = 0;

                        let kilometerCost = 0;

                        let totalRouteCost = 0;

                        const historyApiURL = `http://104.131.12.58/api/get\_history?user\_api\_hash=${user\_api\_hash\_value}&device\_id=${dData.deviceID}&from\_date=${fromDate}&from\_time=${fromTime}&to\_date=${toDate}&to\_time=${toTime}`;

                        const historyResponse = await fetch(historyApiURL);

                        const historyData = await historyResponse.json();

                        totalDistance = historyData.distance\_sum;

                        const HistoryRecordItems = historyData.items;

                        let lastRecordXML = new DOMParser().parseFromString(HistoryRecordItems[0]['items'][0]['other'], "text/xml");

                        if (lastRecordXML.getElementsByTagName('fuelused').length > 0) {

                            let lastFuelRecord = parseInt(lastRecordXML.getElementsByTagName('fuelused')[0].textContent);

                            let initRecord = HistoryRecordItems[HistoryRecordItems.length - 1];

                            let initRecordXML = new DOMParser().parseFromString(initRecord['items'][0]['other'], "text/xml");

                            let initFuelRecord = parseInt(initRecordXML.getElementsByTagName('fuelused')[0].textContent);

                            totalFuelUsed = Math.round((initFuelRecord - lastFuelRecord) \* 0.264172, 1);

                            if (totalFuelUsed === 0) {

                                unitPerformance = "No data";

                                kilometerCost = "No data";

                                totalRouteCost = "No data";

                            } else {

                                distanceValue = parseFloat(totalDistance.replace(/[^\d.]/g, ''));

                                unitPerformance = (totalFuelUsed / distanceValue).toFixed(1);

                                kilometerCost = (totalFuelUsed \* gallonPrice / distanceValue).toFixed(1);

                                totalRouteCost = (totalFuelUsed \* gallonPrice).toFixed(1);

                            }

                        } else {

                            totalFuelUsed = "No data";

                            unitPerformance = "No data";

                            kilometerCost = "No data";

                            totalRouteCost = "No data";

                        }

                        let eachDeviceData = {

                            deviceID: dData.deviceID,

                            driverName: dData.driverName,

                            deviceName: dData.deviceName,

                            totalDistance: totalDistance,

                            totalFuelUsed: totalFuelUsed,

                            unitPerformance: unitPerformance,

                            kilometerCost: kilometerCost,

                            totalRouteCost: totalRouteCost

                        };

                        totalOutputData.push(eachDeviceData);

                    }

#### Result

* HTML

if(outputFormat == "html")

                    {

                        let tbodyOutputData = "";

                        for (let k = 0; k<totalOutputData.length; k++){

                            tbodyOutputData += "<tr>";

                            tbodyOutputData += `<td>${totalOutputData[k].driverName}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].deviceName}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].totalDistance}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].totalFuelUsed}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].unitPerformance}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].kilometerCost}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].totalRouteCost}</td>`;

                            tbodyOutputData += "</tr>";

                        }

                        let tableOutput = `

                        <table class="table table-bordered">

                            <thead>

                                <tr>

                                    <th>Driver Name</th>

                                    <th>Device Name</th>

                                    <th>Total Distance</th>

                                    <th>Total Fuel Used</th>

                                    <th>Unit Performance</th>

                                    <th>Cost per Kilometer</th>

                                    <th>Total Cost</th>

                                </tr>

                            </thead>

                            <tbody>

                            ${tbodyOutputData}

                            </tbody>

                        </table>

                        `

                        GenerateHTML(title, fromDate, fromTime, toDate, toTime, tableOutput);

                    }

* XML

if(outputFormat == "xml"){

                        let xmlOutput = "<root>";

                        for (let k = 0; k<totalOutputData.length; k++){

                            xmlOutput += "<record>";

                            xmlOutput += `<driverName>${totalOutputData[k].driverName}</driverName>`;

                            xmlOutput += `<deviceName>${totalOutputData[k].deviceName}</deviceName>`;

                            xmlOutput += `<totalDistance>${totalOutputData[k].totalDistance}</totalDistance>`;

                            xmlOutput += `<totalFuelUsed>${totalOutputData[k].totalFuelUsed}</totalFuelUsed>`;

                            xmlOutput += `<unitPerformance>${totalOutputData[k].unitPerformance}</unitPerformance>`;

                            xmlOutput += `<kilometerCost>${totalOutputData[k].kilometerCost}</kilometerCost>`;

                            xmlOutput += `<totalRouteCost>${totalOutputData[k].totalRouteCost}</totalRouteCost>`;

                            xmlOutput += "</record>";

                        }

                        xmlOutput += "</root>";

                        GenerateXML(title, fromDate, fromTime, toDate, toTime, xmlOutput)

                    }

* PDF

if(outputFormat == "pdf"){

                        let tbodyOutputData = "";

                        for (let k = 0; k<totalOutputData.length; k++){

                            tbodyOutputData += "<tr>";

                            tbodyOutputData += `<td>${totalOutputData[k].driverName}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].deviceName}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].totalDistance}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].totalFuelUsed}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].unitPerformance}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].kilometerCost}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].totalRouteCost}</td>`;

                            tbodyOutputData += "</tr>";

                        }

                        let tableOutput = `

                        <div class="container">

                        <table class="ptable table-bordered">

                            <thead>

                                <tr>

                                    <th>Driver Name</th>

                                    <th>Device Name</th>

                                    <th>Total Distance</th>

                                    <th>Total Fuel Used</th>

                                    <th>Unit Performance</th>

                                    <th>Cost per Kilometer</th>

                                    <th>Total Cost</th>

                                </tr>

                            </thead>

                            <tbody>

                            ${tbodyOutputData}

                            </tbody>

                        </table>

                        </div>

                        `

                        GeneratePDF(title, fromDate, fromTime, toDate, toTime, tableOutput)

                    }

* CSV

if(outputFormat == "csv"){

                        GenerateCSV(title, fromDate, fromTime, toDate, toTime, totalOutputData)

                    }

### Temperature Report

#### Requirement Research

Teltonika GPS send physical probes or 4 bluetooth wireless sensors

If one of the temperature parameters are not coming then the box should be left blank.

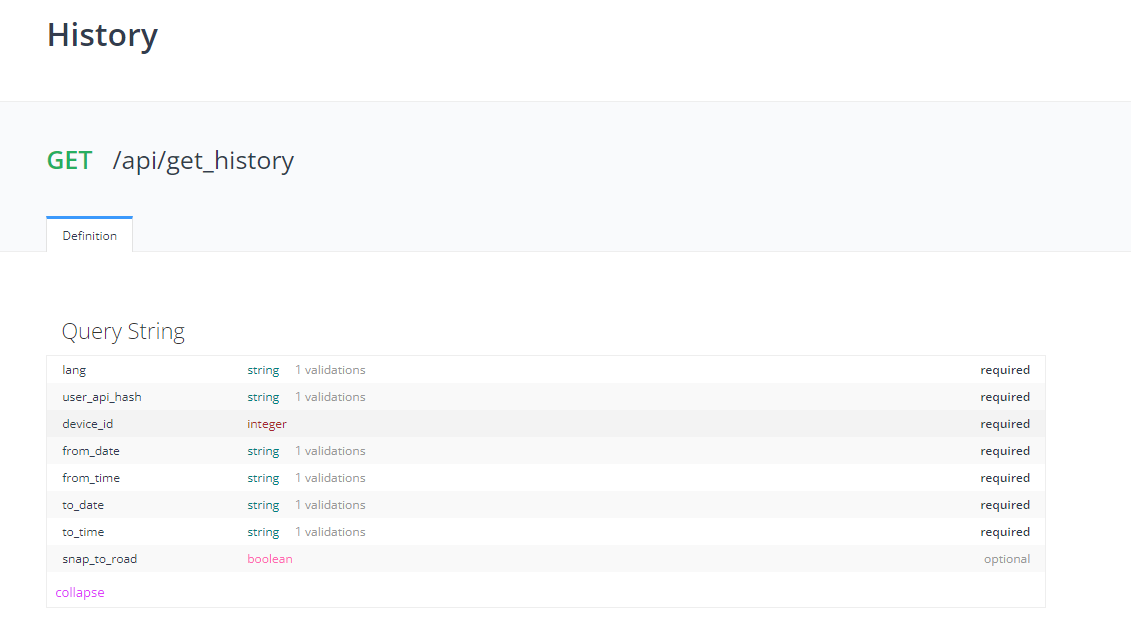
When the Probe is wired, the values arrive under parameters temp1,2,3,4 and when it is Bluetooth it arrives as Temp5,6,7,8

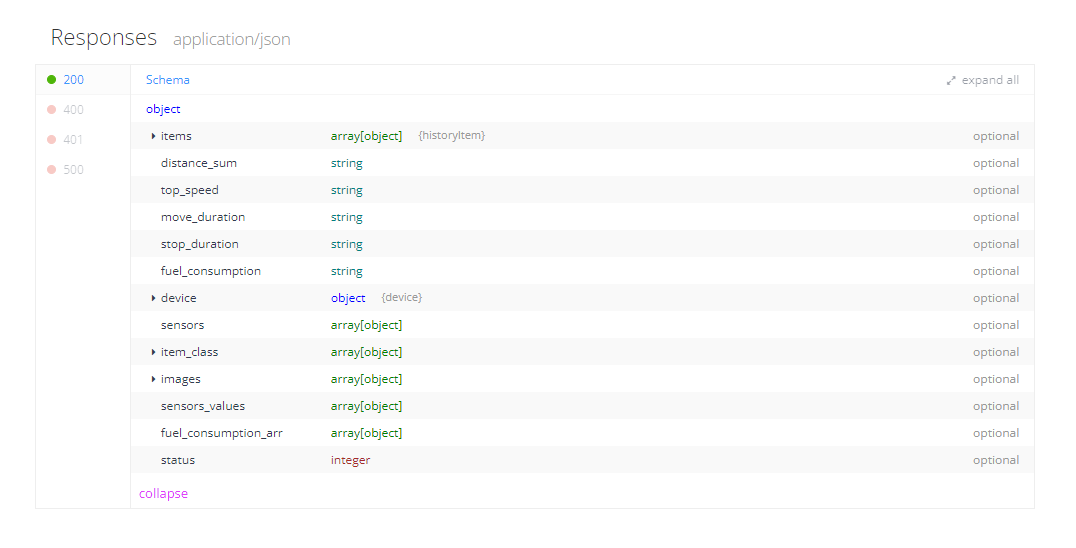
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Device | Date | Temp5/1 | Temp6/2 | Temp7/3 | Temp8/4 |
|  |  | ‘C | ‘C | ‘C | ‘C |

#### API Research

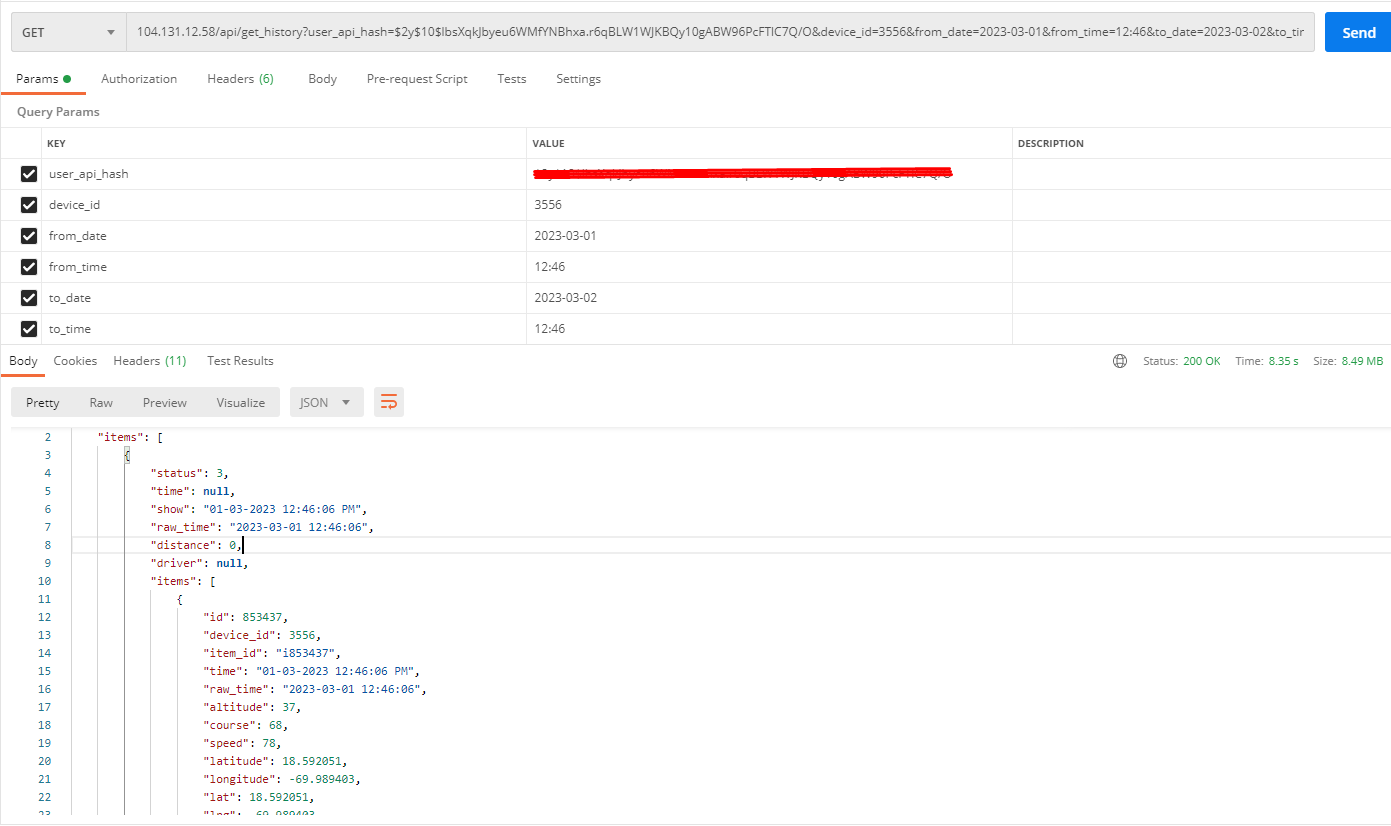
To get temp values, we have to analyze vehicle’s history

<https://gpswox.api-docs.io/v1.0/history/CpN89SKgZDa27nhsi>

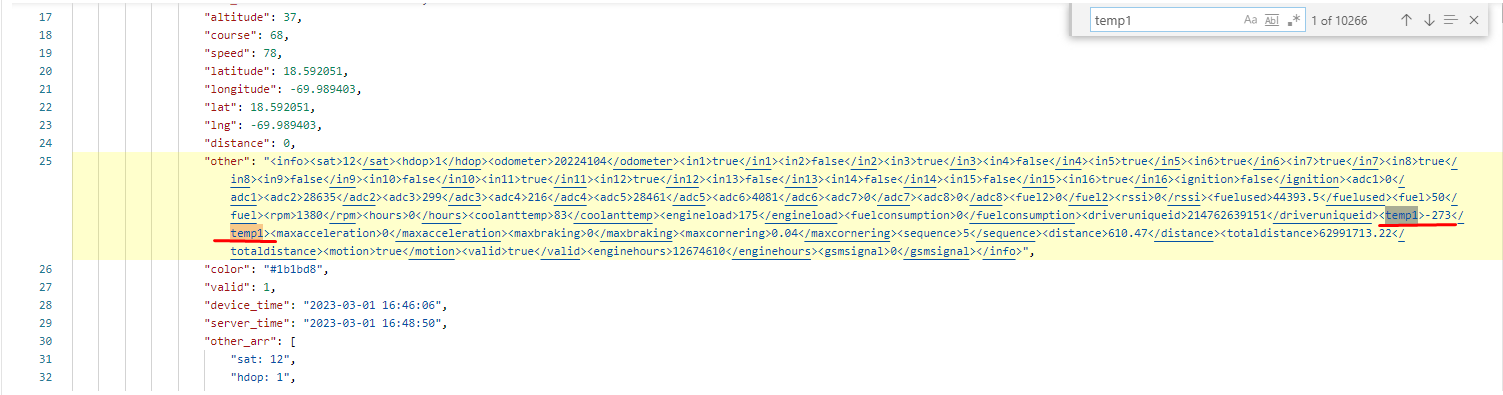




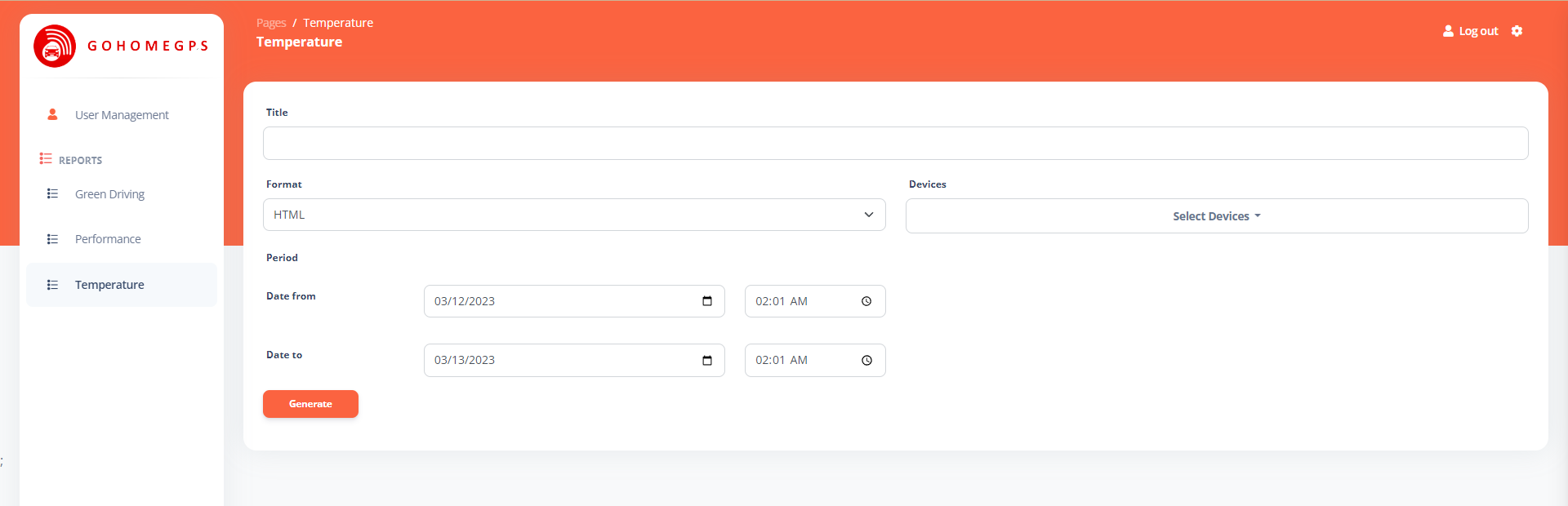
By provide user\_api\_hash, device\_id and period, we can get history of each device.



To get temp value, we have to analyze this part



#### UI Building



#### Implement

* Route

\routes\web.php

    Route::get('/{page}', [PageController::class, 'index'])->name('page');

* Controller

\app\Http\Controllers\PageController.php

public function temperature()

    {

        $response = Http::get('104.131.12.58/api/get\_devices?user\_api\_hash='.session('user\_api\_hash'));

        $jsonData = $response->json();

        $devices = $jsonData[0]['items'];

        return view("pages.temperature", compact('devices'));

    }

* View

\resources\views\pages\temperature.blade.php

* Data Analyze

for (let i = 0; i < devicesData.length; i++) {

                        const dData = JSON.parse(devicesData[i]);

                        let HistoryapiURL = `http://104.131.12.58/api/get\_history?user\_api\_hash=${user\_api\_hash\_value}&device\_id=${dData.deviceID}&from\_date=${fromDate}&from\_time=${fromTime}&to\_date=${toDate}&to\_time=${toTime}`;

                        let HistoryResponse = await fetch(HistoryapiURL, { timeout: 180000 });

                        let HistoryJsonData = await HistoryResponse.json();

                        let prevTemp1 = prevTemp2 = prevTemp3 = prevTemp4 = 0;

                        let curTemp1 = curTemp2 = curTemp3 = curTemp4 = "-";

                        let recordDate = 0;

                        let device\_name = HistoryJsonData.device.name;

                        for (let j = 0; j < HistoryJsonData.items.length; j++) {

                            let HistoryItems = HistoryJsonData.items[j];

                            for (let k = 0; k < HistoryItems.items.length; k++) {

                                let HistoryItem = HistoryItems.items[k];

                                if (HistoryItem.other && HistoryItem.other !== "")

                                {

                                    let parser = new DOMParser();

                                    let xml = parser.parseFromString(HistoryItem.other, "text/xml");

                                    if (xml.getElementsByTagName("temp1")[0]) {

                                        curTemp1 = parseFloat(xml.getElementsByTagName("temp1")[0].textContent);

                                    }

                                    if (xml.getElementsByTagName("temp5")[0]) {

                                        curTemp1 = parseFloat(xml.getElementsByTagName("temp5")[0].textContent);

                                    }

                                    if (xml.getElementsByTagName("temp2")[0]) {

                                        curTemp2 = parseFloat(xml.getElementsByTagName("temp2")[0].textContent);

                                    }

                                    if (xml.getElementsByTagName("temp6")[0]) {

                                        curTemp2 = parseFloat(xml.getElementsByTagName("temp6")[0].textContent);

                                    }

                                    if (xml.getElementsByTagName("temp3")[0]) {

                                        curTemp3 = parseFloat(xml.getElementsByTagName("temp3")[0].textContent);

                                    }

                                    if (xml.getElementsByTagName("temp7")[0]) {

                                        curTemp3 = parseFloat(xml.getElementsByTagName("temp7")[0].textContent);

                                    }

                                    if (xml.getElementsByTagName("temp4")[0]) {

                                        curTemp4 = parseFloat(xml.getElementsByTagName("temp4")[0].textContent);

                                    }

                                    if (xml.getElementsByTagName("temp8")[0]) {

                                        curTemp4 = parseFloat(xml.getElementsByTagName("temp8")[0].textContent);

                                    }

                                    if (prevTemp1 !== curTemp1 || prevTemp2 !== curTemp2 || prevTemp3 !== curTemp3 || prevTemp4 !== curTemp4)

                                    {

                                        prevTemp1 = curTemp1;

                                        prevTemp2 = curTemp2;

                                        prevTemp3 = curTemp3;

                                        prevTemp4 = curTemp4;

                                        recordDate = HistoryItem.raw\_time;

                                        let data = {

                                        deviceName: device\_name,

                                        recordDate: recordDate,

                                        temp1: prevTemp1,

                                        temp2: prevTemp2,

                                        temp3: prevTemp3,

                                        temp4: prevTemp4,

                                        };

                                        totalOutputData.push(data);

                                    }

                                }

                            }

                        }

                    }

We have to parse “other” XML string to get temp value and have to get changed record of each temperature

#### Result

* HTML

if(outputFormat == "html")

                    {

                        let tbodyOutputData = "";

                        for (let k = 0; k<totalOutputData.length; k++){

                            tbodyOutputData += "<tr>";

                            tbodyOutputData += `<td>${totalOutputData[k].deviceName}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].recordDate}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].temp1}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].temp2}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].temp3}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].temp4}</td>`;

                            tbodyOutputData += "</tr>";

                        }

                        let tableOutput = `

                        <table class="table table-bordered">

                            <thead>

                                <tr>

                                    <th>Device Name</th>

                                    <th>Record Date</th>

                                    <th>Temp1</th>

                                    <th>Temp2</th>

                                    <th>Temp3</th>

                                    <th>Temp4</th>

                                </tr>

                            </thead>

                            <tbody>

                            ${tbodyOutputData}

                            </tbody>

                        </table>

                        `

                        GenerateHTML(title, fromDate, fromTime, toDate, toTime, tableOutput);

                    }

* XML

if(outputFormat == "xml"){

                        let xmlOutput = "<root>";

                        for (let k = 0; k<totalOutputData.length; k++){

                            xmlOutput += "<record>";

                            xmlOutput += `<deviceName>${totalOutputData[k].deviceName}</deviceName>`;

                            xmlOutput += `<recordDate>${totalOutputData[k].recordDate}</recordDate>`;

                            xmlOutput += `<temp1>${totalOutputData[k].temp1}</temp1>`;

                            xmlOutput += `<temp2>${totalOutputData[k].temp2}</temp2>`;

                            xmlOutput += `<temp3>${totalOutputData[k].temp3}</temp3>`;

                            xmlOutput += `<temp4>${totalOutputData[k].temp4}</temp4>`;

                            xmlOutput += "</record>";

                        }

                        xmlOutput += "</root>";

                        GenerateXML(title, fromDate, fromTime, toDate, toTime, xmlOutput)

                    }

* PDF

if(outputFormat == "pdf"){

                        let tbodyOutputData = "";

                        for (let k = 0; k<totalOutputData.length; k++){

                            tbodyOutputData += "<tr>";

                            tbodyOutputData += `<td>${totalOutputData[k].deviceName}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].recordDate}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].temp1}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].temp2}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].temp3}</td>`;

                            tbodyOutputData += `<td>${totalOutputData[k].temp4}</td>`;

                            tbodyOutputData += "</tr>";

                        }

                        let tableOutput = `

                        <div class="container">

                        <table class="ptable table-bordered">

                            <thead>

                                <tr>

                                    <th>Device Name</th>

                                    <th>Record Date</th>

                                    <th>Temp1</th>

                                    <th>Temp2</th>

                                    <th>Temp3</th>

                                    <th>Temp4</th>

                                </tr>

                            </thead>

                            <tbody>

                            ${tbodyOutputData}

                            </tbody>

                        </table>

                        </div>

                        `

                        GeneratePDF(title, fromDate, fromTime, toDate, toTime, tableOutput)

                    }

* CSV

if(outputFormat == "csv"){

                        GenerateCSV(title, fromDate, fromTime, toDate, toTime, totalOutputData)

                    }

## Generate Report Result

### HTML

From inputted title and generated data, we can generate report as HTML format

function GenerateHTML(title, fromDate, fromTime, toDate, toTime, tableOutput){

            const htmlString = `

            <html>

                <head>

                    <title>${title}</title>

                    <style>

                        body{

                            font-family: "Helvetica Neue",Helvetica,Arial,sans-serif;

                            font-size: 14px;

                            line-height: 1.42857143;

                            color: #333;

                        }

                        .container{

                            width: 80%;

                            padding-right: 15px;

                            padding-left: 15px;

                            margin-right: auto;

                            margin-left: auto;

                            text-align: center;

                        }

                        .table{

                            border: 1px solid #ddd;

                            width: 100%;

                            max-width: 100%;

                            margin-bottom: 20px;

                            background-color: transparent;

                            border-collapse: collapse;

                            border-spacing: 0;

                        }

                        thead{

                            display: table-header-group;

                            vertical-align: middle;

                            border-color: inherit;

                            border-collapse: collapse;

                            border-spacing: 0;

                        }

                        .table-bordered>thead>tr>th{

                            border-top: 0;

                            border-bottom-width: 2px;

                            border: 1px solid #ddd;

                            padding: 8px;

                            line-height: 1.42857143;

                        }

                        .table>thead>tr>th{

                            vertical-align: bottom;

                        }

                        th{

                            text-align: left;

                        }

                        .table-bordered>tbody>tr>td {

                            border: 1px solid #ddd;

                        }

                        .table>tbody>tr>td{

                            padding: 8px;

                            line-height: 1.42857143;

                            vertical-align: top;

                        }

                        .ptable{

                            border: 1px solid #ddd;

                            width: 100%;

                            max-width: 100%;

                            margin-bottom: 20px;

                            background-color: transparent;

                            border-collapse: collapse;

                            border-spacing: 0;

                        }

                        .ptable>tbody>tr>td{

                            padding: 8px;

                            line-height: 1.42857143;

                            vertical-align: top;

                        }

                    </style>

                </head>

                <body>

                    <div class="container">

                        <h1>${title}</h1>

                        <p>From ${fromDate} ${fromTime} To ${toDate} ${toTime}</p>

                        ${tableOutput}

                    </div>

                </body>

            </html>

            `;

            // Convert the HTML string to a Blob

            const htmlBlob = new Blob([htmlString], {type: 'text/html'});

            // Create a URL for the Blob

            const url = URL.createObjectURL(htmlBlob);

            // Create a link element to download the HTML file

            const link = document.createElement('a');

            link.href = url;

            link.download = `${title}${fromDate}${fromTime}-${toDate}${toTime}.html`;

            // Append the link to the document

            document.body.appendChild(link);

            // Click the link to download the file

            link.click();

            // Clean up the URL object

            URL.revokeObjectURL(url);

        }

### XML

function GenerateXML(title, fromDate, fromTime, toDate, toTime, xmlOutput){

            const xmlVersion = <?php echo json\_encode('<?xml version="1.0" encoding="UTF-8"?>'); ?>;

            const xmlString = `${xmlVersion}

            ${xmlOutput}

            `;

            // Convert the XML string to a Blob

            const blob = new Blob([xmlString], { type: 'text/xml' });

            // Create a URL for the Blob

            const url = URL.createObjectURL(blob);

            // Create a link element to download the HTML file

            const link = document.createElement('a');

            link.href = url;

            link.download = `${title}${fromDate}${fromTime}-${toDate}${toTime}.xml`;

            // Append the link to the document

            document.body.appendChild(link);

            // Click the link to download the file

            link.click();

            // Clean up the URL object

            URL.revokeObjectURL(url);

        }

### PDF

function GeneratePDF(title, fromDate, fromTime, toDate, toTime, htmlOutput){

            var opt = {

                margin: 1,

                filename: `${title}${fromDate}${fromTime}-${toDate}${toTime}.pdf`,

                image: {

                type: 'jpeg',

                quality: 0.98

                },

                html2canvas: {

                scale: 0.7

                },

                jsPDF: {

                unit: 'in',

                format: 'letter',

                orientation: 'landscape',

                pageSize : 'A4'

                }

            };

            // Convert the HTML to PDF using the html2pdf library

            html2pdf().set(opt).from(htmlOutput).save();

        }

### CSV

function GenerateCSV(title, fromDate, fromTime, toDate, toTime, data){

            const titleKeys = Object.keys(data[0]);

            const refinedData = [];

            refinedData.push(titleKeys);

            console.log(data)

            data.forEach(item => {

                refinedData.push(Object.values(item));

            })

            let csvContent = ''

            refinedData.forEach(row => {

            csvContent += row.join(',') + '\n'

            })

            const blob = new Blob([csvContent], { type: 'text/csv;charset=utf-8,' })

            const objUrl = URL.createObjectURL(blob)

            const link = document.createElement('a')

            link.href = objUrl;

            link.download = `${title}${fromDate}${fromTime}-${toDate}${toTime}.csv`;

            // Append the link to the document

            document.body.appendChild(link);

            // Click the link to download the file

            link.click();

            // Clean up the URL object

            URL.revokeObjectURL(objUrl);

        }

# Step 2:

## User Management System

Include an option to create and manage users, who must be able to connect to the GPSWOX server in order to login both to said server and to the new Cpanel.

### User enable/Disable

### User info edit

### User permission change

### User delete