EE3L11

Project SUNRISE: Dashboard en Communicatie voor een e-bike oplaadstation

BAP Team D, Groep Server

Delft University of Technology

Nourdin Aït el Mehdi, 4276825
Timothy de Moor, 4272536

15 juni 2016
Version 1.0

TU Delft Delft University of Technology

SUNRISE

Challenge the future

Samenvatting

In deze thesis wordt het ontwerpproces van het Bachelor Afstudeer Project beschreven. De bedoeling van dit project is om een softwaresysteem te maken voor een oplaadstation voor elektrische fietsen. Het speciale aan dit oplaadstation is dat de fietsen voornamelijk opgeladen worden door energie opgewekt door zonnepanelen. Een gebruiker moet bij aankomst bij het oplaadstation een oplaadpunt reserveren via een website.

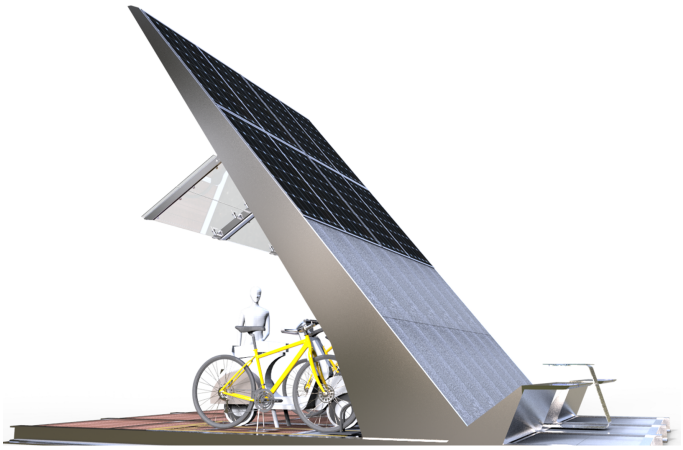
Het oplaadstation dient ook als proeftuin. Daarom bevat het oplaadstation verschillende meetinstrumenten om gegevens over onder andere het weer te verzamelen. Deze data wordt opgevraagd door een minicomputer (ODROID) die in dit oplaadstation is ingebouwd. De data moet worden opgestuurd naar de server van de TU Delft. Daar moet deze data opgeslagen worden in een database. De beheerder van het station moet de data op kunnen vragen uit de database via een speciale webpagina. In dit verslag wordt specifiek ingegaan op de webpagina voor de beheerder, de communicatie met de database, en de communicatie met de ODROID in het station.

Alle eisen die gesteld waren aan het project voor groep server zijn vervuld. De beheerder pagina (het dashboard) kan zelfs meer dan is opgegeven. Het dashboard past zichzelf automatisch aan bij een verandering in de database. De data kan op het dashboard zichtbaar gemaakt worden in grafieken en of tabellen. De communicatie tussen de server en de database is beveiligd zodat kwaadwillenden niet kunnen inbreken op de database. De ODROID kan succesvol data opsturen en verkrijgen, en de server zet of haalt deze data in of uit de corresponderende tabel in de database.

Inhoudsopgave

1 Introductie

Dit project is in opdracht van de TU Delft. De opdracht bestaat uit het maken van een softwaresysteem voor een oplaadstation voor elektrische fietsen (e-bikes). Dit oplaadstation zal op de campus van de TU Delft worden gebouwd. In figuur 1 is een conceptversie van het oplaadstation te zien. Dit is een 5 meter hoge fietsenstalling waarop 8 zonnepanelen geplaatst zijn. Deze zonnepanelen laden de batterijen op die zich in de fietsenstalling bevinden. De fietsenstalling bevat 6 oplaadpunten om een e-bike op te laden. Het opladen kan zowel bedraad als draadloos afhankelijk van of de e-bike dit ondersteund. Naast de batterijen en zonnepanelen bevat het oplaadstation veel meetapparatuur. Er zit een weerstation boven op het oplaadstation gemonteerd die de hele dag door verschillende metingen doet. Daarnaast word met temperatuursensoren de temperatuur van de zonnepanelen in de gaten gehouden.



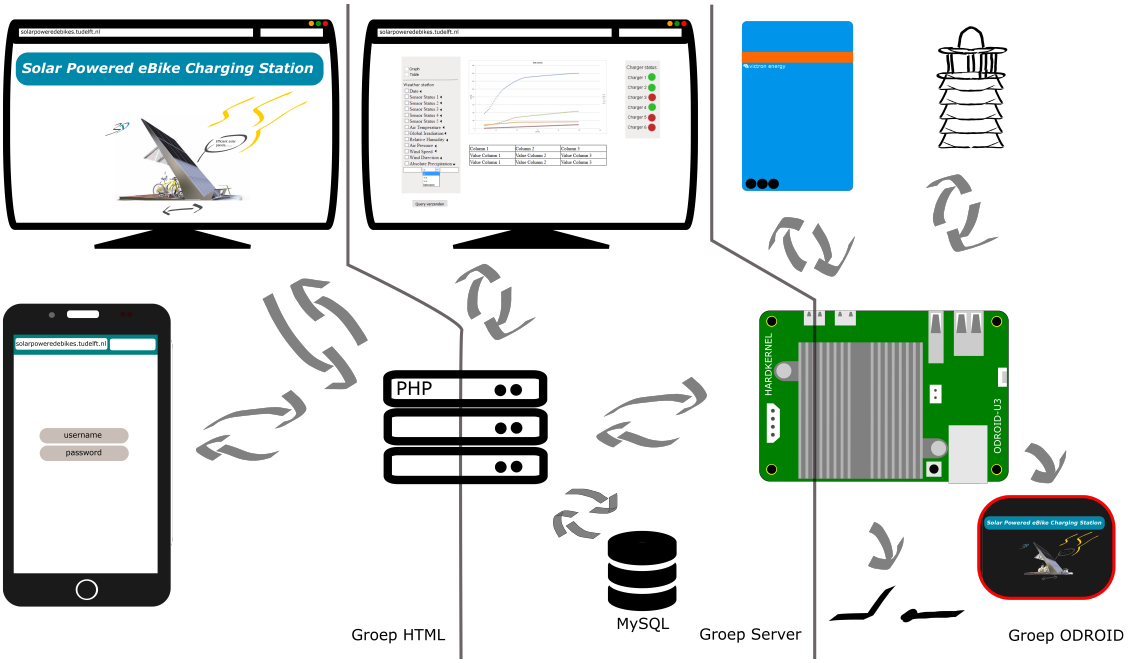
Figuur 1: Concept van het oplaadstation.

Het SUNRISE project (Smart Unified Networking Rig for an Integrated Solar Ebike charger) is voorgesteld als bachelor eindproject. In figuur 2 is het systeem overzicht van het project te zien. Het resultaat van het project moet er voor zorgen dat de gebruikers bij aankomst bij de fietsenstalling een oplaadpunt kunnen reserveren. Het reserveren moet gaan via een webpagina waarop de gebruiker inlogt via zijn telefoon. Hiervoor moet de website verbinding maken met een server. Deze server zal zorgen voor de nodige functionaliteit om te communiceren met de rest van het systeem. Naast een webpagina om te reserveren moet er een informatieve pagina voor de media gemaakt worden.
Het oplaadstation wordt onder andere gebruikt als proeftuin en bevat daarom veel elektronica die vooral metingen doen. Alle data van deze elektronica moet opgeslagen worden in een database. Maar zo een database is niet overzichtelijk en al helemaal niet gebruiksvriendelijk. Daarvoor moet er een speciale website worden gemaakt, het “dashboard”.
Om de data van de meetapparatuur op de server te krijgen is een minicomputer (ODROID) in het oplaadstation ingebouwd. Door middel van een internetverbinding heeft die toegang tot de server. De ODROID zorgt er onder andere voor dat een vooraf gereserveerd oplaadpunt van stroom wordt voorzien. Als laatst moet de ODROID een scherm aansturen die in het oplaadstation gebouwd wordt. Het scherm zal actuele informatie over het oplaadstation zichtbaar maken.

1.1 Probleemstelling

Groep Server van het afstudeerproject focust zich voornamelijk op de communicatie tussen de server en de database en op de communicatie tussen de server en de ODROID. Ook moet een website gemaakt worden waarop de metingen van de meetinstrumenten overzichtelijk te zien zijn.

De precieze eisen staan genoemd in sectie 2.



Figuur 2: Een schematische weergave van het project

1.2 State-of-the-art analyse

Voordat werd besloten dat dit een Bachelor Afstudeer Project zou worden, waren er al mensen die aan het project werkten. Hierdoor was er al code beschikbaar. Deze code bevatte methodes om de meetinstrumenten uit te lezen en de data op te sturen. Ook was er voor de server code beschikbaar die de data in de database kon zetten. Deze code was echter nog niet getest met alle meetsystemen. Het gehele oplaadstation met aansturingselektronica was al ontwikkeld en wordt beschikbaar gesteld. Ook is een webserver beschikbaar gesteld die een PHP programmeeromgeving en een phpMyAdmin database bevat [1].

Voor de JavaScript code wordt gebruik gemaakt van “jQuery” [2]. Dit is een bibliotheek die het gebruik van JavaScript samen met HTML efficiënter maakt. Voor de grafieken wordt de online beschikbare bibliotheek “Chart.js” gebruikt [3]. Dit is een open-source JavaScript bibliotheek gemaakt door de community. Deze bibliotheek versnelt het maken van grafieken op een canvas. Verder zijn er nog drie bibliotheken gebruikt: “Moment.js” [4], “Timepicker.js” [5] en “TableToCSV.js” [6]. Deze drie bibliotheken zijn beschikbaar gesteld onder de MIT licentie.Voor alle andere functionaliteiten wordt gebruik gemaakt van functies die standaard beschikbaar zijn in de verschillende programmeertalen.

1.3 Overzicht

Dit verslag beschrijft het ontwikkelproces van een deel van het softwaresysteem voor het oplaadstation. Dat houdt in dat de volgende paragrafen zullen gaan over de server, het dashboard en de communicatie tussen de server en de ODROID. Als eerst worden de eisen die gesteld zijn door de opdrachtgever beschreven (Sectie 2). Daarna volgt het ontwerpproces (Sectie 3). Hierin worden de gemaakte keuzes verantwoord. Hierna volgen de resultaten van het project en de testresultaten

van de communicatie tussen de server en de ODROID (Sectie 4). Als laatst volgen de discussie (Sectie 5) en de conclusie (Sectie 6).

2 Programma van eisen

Het uiteindelijke doel is om een server te hebben waardoor alle delen van het systeem met elkaar kunnen communiceren. De ODROID in het oplaadstation moet alle data van de meetsystemen doorsturen naar de server. De server zal verder ervoor zorgen dat die data opgeslagen wordt in een database. Daarnaast moet die data ook zichtbaar gemaakt kunnen worden. Hiervoor moet een dashboardpagina gemaakt worden. Hieronder staat een lijst van de eisen voor respectievelijk de server, het dashboard en de ODROID.

Server

[1.1] De code op de server moet in PHP en MySQL geschreven worden;

[1.2] De server moet kunnen communiceren met de database;

[1.3] De server moet een pagina bevatten waarop alle data in de database op te vragen is, de zogeheten dashboardpagina.

Dashboard

[2.1] Op het dashboard moet de huidige status van de laders zichtbaar zijn en de status moet handmatig aan te passen zijn;

[2.2] De code voor de website zelf moet in HTML5, CSS en JavaScript geschreven worden;

[2.3] De data vanuit de database moet uitgelezen kunnen worden op het dashboard;

[2.4] De grafieken moeten met behulp van HTML5 canvas gemaakt worden;

[2.5] Om de pagina te bereiken moet men eerst inloggen.

ODROID

[3.1] De ODROID moet meetdata kunnen opsturen naar de server;

[3.2] Als de connectie met het internet weg valt, moet deze meetdata gebufferd worden om later alsnog op te sturen;

[3.3] Als de buffer vol is, dan vervallen de eerste waardes om plaats te maken voor nieuwe;

[3.4] De ODROID moet aan de server kunnen laten weten wanneer een fiets losgekoppeld is, zodat de reserveerpagina deze plekken weer kan aanbieden;

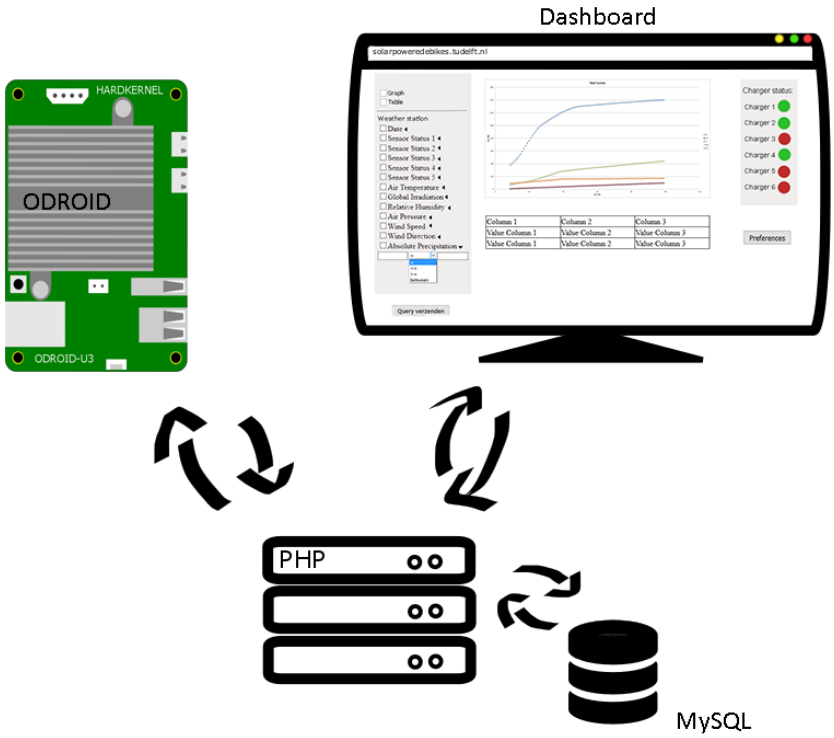
[3.5] De ODROID moet de huidige status van de laders kunnen opvragen, zodat de correcte laders stroom kunnen krijgen;

[3.6] De code op de ODROID mag geen geheugenlekken bevatten en moet altijd blijven werken;

[3.7] De code op de ODROID moet in C geschreven worden.

3 Het ontwerpproces

Dit hoofdstuk beschrijft het volledige ontwerpproces van de onderdelen te zien in figuur 3. In het midden van het systeemoverzicht is de server te zien. De server zorgt ervoor dat alle onderdelen van het systeem met elkaar verbonden kunnen worden. De TU Delft heeft voor dit project een server beschikbaar gesteld. De toegang tot deze server is beschermd door middel van een inlogsysteem. Op deze server staat ook al een database van phpMyAdmin [1] die gebruik maakt van MySQL. Dit is een gestandaardiseerde taal die gebruikt wordt om te communiceren met de database. Op de server word naast MySQL de programmeertaal PHP gebruikt [1.1]. Er is gekozen voor PHP omdat de code die daarin geschreven wordt bij het opvragen van een webpagina op de server blijft staan. Het voordeel hiervan is dat kwaadwillenden geen toegang hebben tot de implementatie van de code. Verder worden de talen HTML, CSS en jQuery naast PHP gebruikt voor het maken van het dashboard. De code die in die drie talen geschreven is, kan in de browser wel zichtbaar gemaakt worden.



Figuur 3: Het systeemoverzicht

Rechtsboven in het systeemoverzicht is het dashboard te zien. Het dashboard is voornamelijk voor de beheerder en moet daarom ook beschermd worden door onder andere een wachtwoord [2.5]. Maar naast de administrator zal de pagina door nog meer andere gebruikers gebruikt worden, daarom word er in de rest van de tekst gesproken over de gebruikers. Op het dashboard moet de data van het oplaadstation te vinden zijn [2.3]. Er is voor gekozen om de data voornamelijk in grafieken en tabellen zichtbaar te maken omdat het vooral gaat om numerieke data. Na dit voorgedragen te hebben aan de opdrachtgever is hier eis [2.4] bij gekomen.

Als laatste moet er ook voor gezorgd worden dat de ODROID kan communiceren met de server.

De eisen [3.1] tot en met [3.7] zijn hierop van toepassing.

3.1 Server

De server is de basis van alle communicatie. De code voor de server gemaakt door de voorgangers die werkten aan dit project is niet gebruikt, omdat deze niet veilig was. Bij gebruik van die code was de database totaal onbeschermd tegen kwaadwillenden.Deze kwaadwillenden konden de gehele database verwijderen zonder enige moeite. Daarnaast was alle data die opgestuurd werd aftelezen in de URL. Op de server is verder alle nodige software aanwezig. Maar voordat alle elementen van het systeem met elkaar kunnen communiceren moeten er een paar problemen worden opgelost:

- Communicatie tussen de server en de database.
- Het beschermen van de server en de database tegen kwaadwillenden.

3.1.1 Beveiliging en communicatie

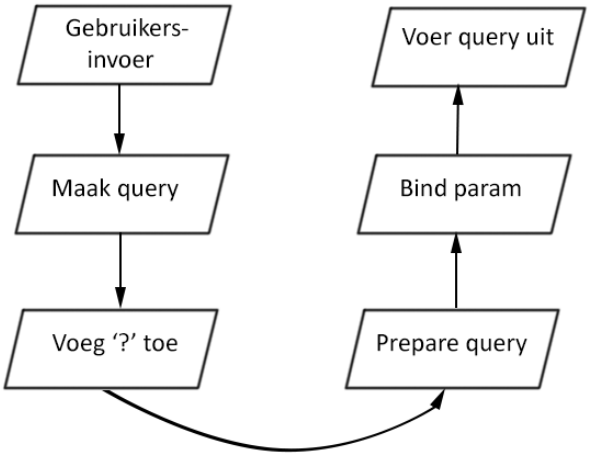
De communicatie tussen de server en database verloopt door middel van PHP. PHP bevat de nodige functies om te communiceren met de database. De database is opgebouwd uit een aantal tabellen die elk weer een aantal kolommen bevatten. In deze kolommen is de data te vinden. Alle handelingen op de database kunnen gedaan worden door middel van SQL. Een instructie die gegeven word aan de database in de vorm van SQL wordt een query genoemd. Er bestaan verschillende soorten query's. In ons geval maken we gebruik van query's om de data aan te passen, toe te voegen of op te halen.

De database is een gevoelig doelwit voor kwaadwillenden omdat door een simpele query de gehele database verwijderd of aangepast kan worden. Daarom is ervoor gekozen de database te beveiligen tegen zulke aanvallen. Zo een aanval word een SQL-injectieaanval genoemd [7]. Op het moment dat er op een website gevraagd word naar invoer van de gebruiker bijvoorbeeld door middel van een invoerveld om data op te vragen van de database, loopt deze het gevaar aangevallen te worden. Zo een aanval maakt gebruik van het feit dat de invoer gegeven door de gebruiker direct in de query word ingevuld. In dit verslag zal er niet ingegaan worden op het gebruik van SQL. Voor meer informatie zie de MySQL handleiding [8]. Een kwaadwillende kan al met een basiskennis van SQL veel schade aanrichten. In plaats van invoer te geven zoals de gemiddelde gebruiker dat zou doen voert een kwaadwillende een SQL-statement in met als doel de database aan te spreken zonder toestemming.

Implementatie
Het beschermen tegen deze aanval kan op meerdere manieren. Twee manieren bleken het meest praktisch. De eerste manier is gebruik maken van de speciale MySQL improved functies van PHP [9]. De tweede manier is de invoer van de gebruiker te beperken en te controleren. Er word gebruik gemaakt van drie soorten query's en dus moeten er drie verschillende functies worden gemaakt. Het zou ook in één functie kunnen, maar hier is niet voor gekozen omdat dit een hele complexe functie tot gevolg zou hebben. Alle delen van het systeem maken gebruik van minimaal één van deze functies.

Voor de query's om data aan te passen of toe te voegen is er gekozen voor de eerste manier. Omdat deze query's simpel opgebouwd zijn en niet bestaan uit door de gebruiker ingevoerde tekst. Maar toch is het nodig om deze query's te beschermen. Deze query's maken gebruik van data opgestuurd van de website via een POST. Deze POST's kunnen door kwaadwillenden zelf gegenereerd worden daarom is er dus toch bescherming nodig. Hiervoor zijn twee functies gemaakt: insertQuery en updateQuery. Beide functies zijn opgebouwd zoals te zien in figuur 4. Het idee van de speciale PHP-functies is om de invoer van de gebruiker altijd te converteren naar een string formaat. Hierdoor kan de invoer van een kwaadwillende die een query meegeeft nooit

uitgevoerd worden. In plaats daarvan zal die invoer opgeslagen worden in de database als een string. De invoer van een gebruiker moet op een bepaalde plaats in de query gezet worden. Deze plaats moet van tevoren worden aangegeven door middel van vraagtekens zoals te zien in figuur 4. Na het toevoegen van de vraagtekens aan de query wordt de query meegeven aan de speciale PHP-functie "prepare query". Deze functie maakt de query klaar om door te geven aan de "bind param" functie. Dit is de functie die er uiteindelijk voor zorgt dat de gebruikersinvoer die omgezet is in een string op de juiste plek terecht komt. Op dit moment is de query klaar om uitgevoerd te worden in de database. Bij een fout in de query wordt er een foutmelding terug gegeven.



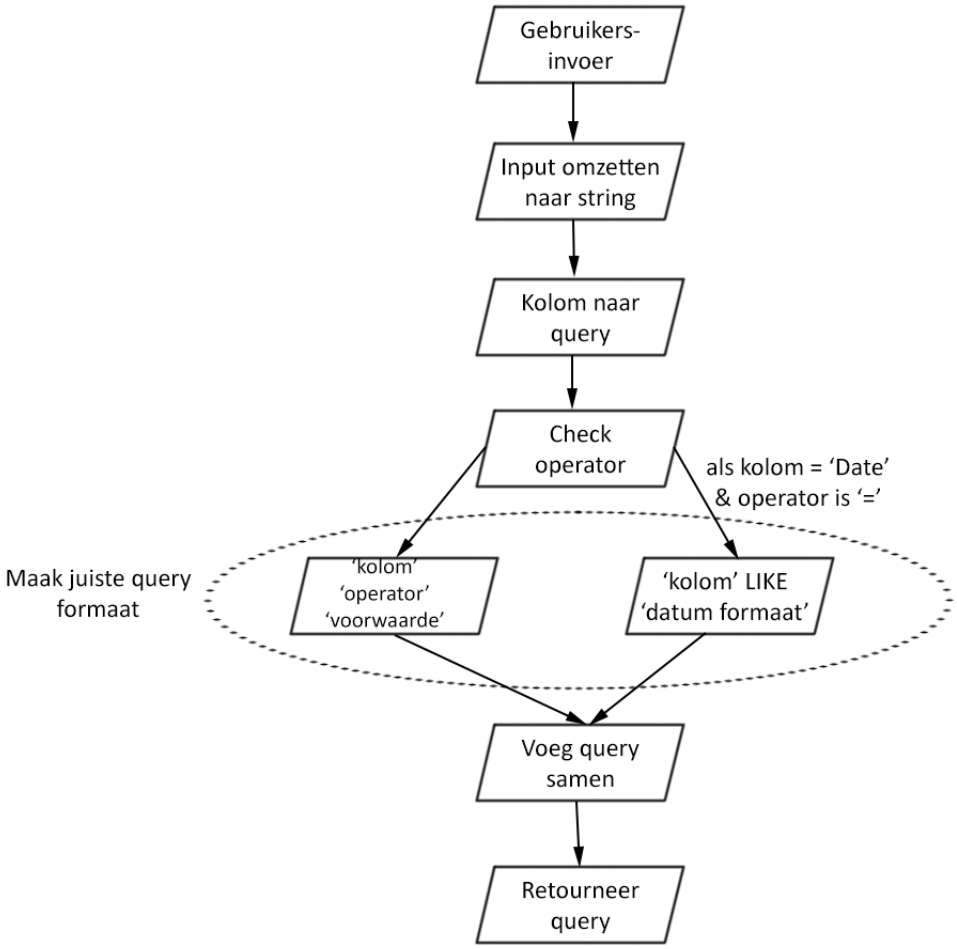
Figuur 4: Het functie overzicht van de insert en update

De meest complexe query is de query om data op te halen omdat deze query veel verschillende parameters bevat. De belangrijkste parameters bestaan uit de operatoren: '<' , '>' , '=' en 'BETWEEN'. Waarbij 'BETWEEN' een speciale SQL operator is die aangeeft dat alle data tussen twee uitgekozen waardes opgehaald moet worden. Deze operatoren zorgen ervoor dat er voorwaarden gesteld kunnen worden aan welke data opgehaald moet worden. Daarnaast bevat de query de tabelnaam en kolomnamen. Voor het maken van deze functie is gekozen voor de tweede manier om de code veilig te maken, omdat het niet mogelijk is om op de plaats van de vraagtekens de juiste waarde te krijgen. Dit komt doordat de plaatsing van de vraagtekens in dit geval te verschillend is. Hierdoor kan de functie "bind param" zijn werk niet op de juiste manier doen.

De functie gemaakt voor het ophalen van de data heet "getQuery". Deze functie kan op twee manieren werken. Er kan direct een query als string meegeven worden of aan de hand van variabelen. Het direct meegeven word gebruikt in het geval dat de gebruiker van het dashboard geen invloed heeft op de inhoud van de query. Deze query word direct naar de database gestuurd en uitgevoerd. In het tweede geval moet de query nog gemaakt worden aan de hand van de waardes in de variabelen. Deze waarden zijn meegegeven door de gebruiker aan de hand van het checkboxmenu uit paragraaf 3.2.1. Hierdoor is de invoer van de gebruiker al beperkt maar toch kan dit ook weer omzeild worden door middel van nagemaakte POST's. Daarom is het nodig om de invoer te controleren. De getQuery functie roept in het geval van manier 2 intern de functie getQueryPrep op. Deze functie zorgt ervoor dat de invoer van de gebruiker gecontroleerd wordt en wordt omgezet in een query.

In figuur 5 is het overzicht van de functie getQueryPrep te zien. De invoer van de gebruiker bestaat uit een tabel, kolommen, voorwaarden en operatoren. Al deze waardes worden omgezet naar string formaat. Daarna worden de kolomnamen op de juiste plek in de query gezet. Voordat

de voorwaarden en operatoren in de query gezet kunnen worden moet er eerst gecontroleerd worden voor welke operator er is gekozen. Dit heeft namelijk invloed op de manier waarop de voorwaarden en operatoren in de query gezet moeten worden. Na het controleren van de operator word er onderscheid gemaakt tussen een normale kolom en de kolom die datums bevat, omdat in het geval van een datum het formaat van de query totaal veranderd. In het geval van een datum word de '=' operator in de functie vervangen door de SQL-operator 'LIKE'. Daarna wordt alles samengevoegd tot één complete query die daarna geretourneerd wordt. Deze query word door de functie getQuery naar de database gestuurd en uitgevoerd. De functie getQuery bestaat uit MySQLi-functies die zorgen voor het verzenden van de query en het ontvangen van het resultaat. Door het gebruik van de functies insertQuery, updateQuery en getQuery word er gecommuniceerd met de database [1.2].



Figuur 5: Het overzicht van de getQueryPrep functie

3.2 Dashboard

Het dashboard bestaat uit veel losse componenten die samen een geheel vormen. De voornaamste taak van het dashboard is om alle data gemeten door het oplaadstation overzichtelijk weer te geven. Dit moet op een gebruiksvriendelijke manier, zodat ook mensen die geen kennis hebben van SQL de data van het oplaadstation kunnen bekijken. Het uiteindelijke dashboard moet voldoen aan de eisen [2.1] tot en met [2.5]. De basisproblemen die opgelost moeten worden zijn:

- Database aanspreken vanaf het dashboard;
- Communicatie tussen de server en het dashboard;
- Het maken van tabellen en grafieken;
- De pagina beveiligen.

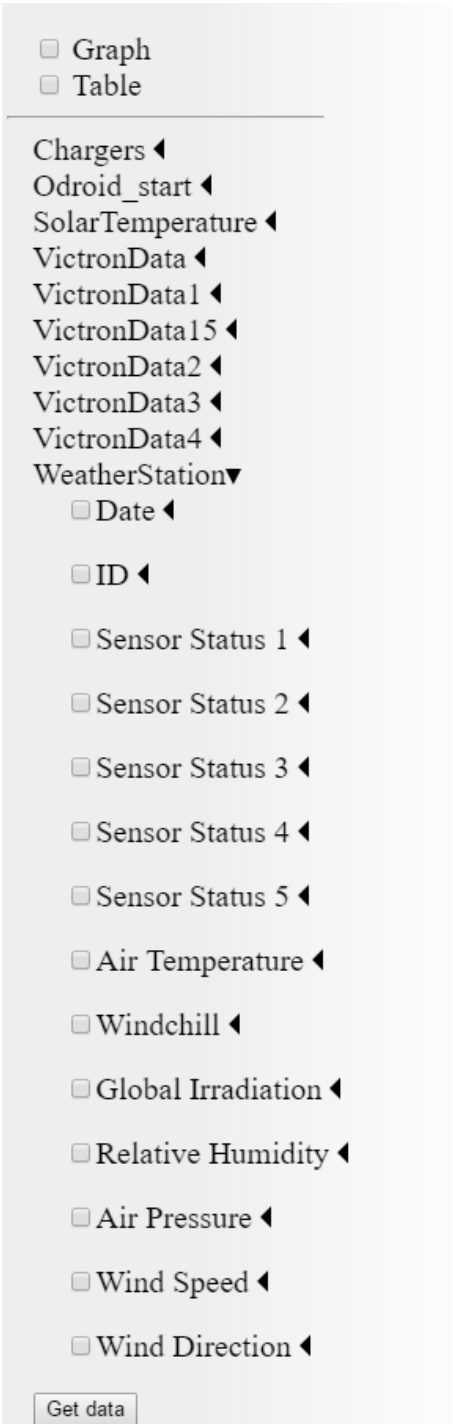
3.2.1 Checkboxmenu

De opdrachtgever had geen eisen gesteld aan het specifieke ontwerp van de dashboardpagina. Daardoor kon het dashboard vrij ontworpen worden. Er is gebruik gemaakt van de programmeertalen HTML5, CSS, jQuery en PHP [2.2]. Voor het ontwerpen van het dashboard is er gekeken naar vergelijkbare websites [10], [11], [12]. Hierbij ging het vooral om hoe de gebruikers op de websites keuzes konden maken om de inhoud van de pagina te filteren. De meeste pagina's maken gebruik van een menu aan de linkerkant van het scherm waarin de keuzes aangevinkt kunnen worden. Daarom is gekozen om op het dashboard een soortgelijk checkboxmenu te implementeren. Dit menu is te zien in figuur 6.

Via dit menu kan de gebruiker filteren op welke data uit de database opgehaald moet worden. Omdat er veel keuzes gemaakt kunnen worden is ervoor gekozen om gebruik te maken van dropdown menu's. Dit zijn menu's die hun inhoud pas zichtbaar maken op het moment dat er ergens op geklikt is. Hierdoor staat niet het hele scherm vol met mogelijke keuzes en blijft het dashboard dus heel overzichtelijk voor de gebruiker. Voor het aanspreken van de database wordt er gebruik gemaakt van de functies gemaakt in paragraaf 3.1.1

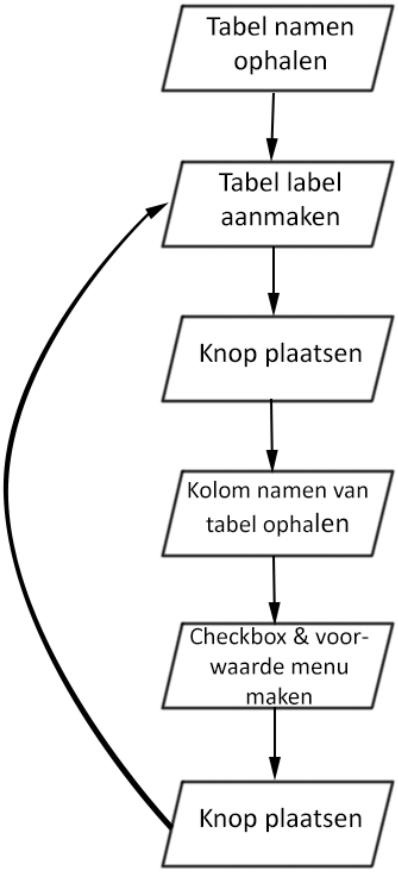
Implementatie
In het implementeren van het checkboxmenu is er veel verder gegaan dan het gewoon maken van een menu. Er is gekozen om ervoor te zorgen dat het gehele menu automatisch gegenereerd wordt aan de hand van alle tabellen en kolommen in de database. Dus de webpagina is niet hardcoded. Hardcoded houdt in dat elke tabelnaam, dropdownmenu enzovoort die te zien is in het checkboxmenu volledig in code is uitgeschreven. Het voordeel van het automatisch genereren van het checkboxmenu is dat de code van het dashboard niet herschreven hoeft te worden als de structuur van de database verandert, dat wil zeggen nieuwe tabellen of kolommen toegevoegd worden. De verandering in de database zullen automatisch doorgevoerd worden in het checkboxmenu. Het nadeel hiervan is dat de code veel complexer wordt en dus meer tijd vergt om te maken.

In figuur 7 is een zeer vereenvoudigd overzicht te zien van de code. Als eerst worden de tabelnamen met behulp van de getQuery functie opgevraagd. Daarna wordt de code per tabel doorlopen. Er wordt een label met de tabel naam aangemaakt. Hierdoor is de tabelnaam in het checkboxmenu te zien. Daarna wordt het knopje waarmee het dropdown menu geopend kan worden achter de naam geplaatst. Elke tabelnaam heeft een andere lengte en dus moet de plaatsing van het knopje zich aanpassen aan de lengte van de tabelnaam. Hierna moeten de kolomnamen van de tabel opgehaald worden. Deze kolomnamen worden gebruikt om de checkboxes en voorwaardenmenu's te maken. De voorwaardenmenu's werken op de zelfde manier als de dropdownmenu's. Voor het openen van het voorwaardenmenu is er weer een knop nodig. Deze knop moet zich ook weer aan kunnen passen aan de lengte van de kolomnaam en wordt als laatste



Figuur 6: Het checkboxmenu.

geplaatst. Dit stuk code wordt telkens opnieuw uitgevoerd totdat alle tabellen uit de database verwerkt zijn.

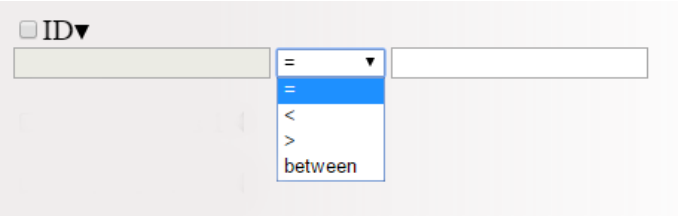


Figuur 7: Het functie overzicht van het checkboxmenu

3.2.2 Voorwaardenmenu

Naast het kiezen van welke data opgehaald moet worden van de database moet het voor de gebruiker ook mogelijk zijn om voorwaarden mee te geven. In figuur 8 is het voorwaardenmenu te zien. De mogelijke voorwaarden bestaan uit getallen of een datum formaat. Dit is zo gedaan zodat de gebruiker specifieke waarden op kan vragen uit de database. Naast deze getallen en datums moet er ook een operator gekozen kunnen worden. De operatoren uit paragraaf 3.1 zijn van te voren vastgesteld. Hierdoor is het makkelijker om de query te maken. Als de gebruiker vrij was om zelf elke gewenste operator in te vullen zou het te complex worden om een functie zoals getQueryPrep te maken. Deze operatoren worden gekozen door het openen van de selectielijst.

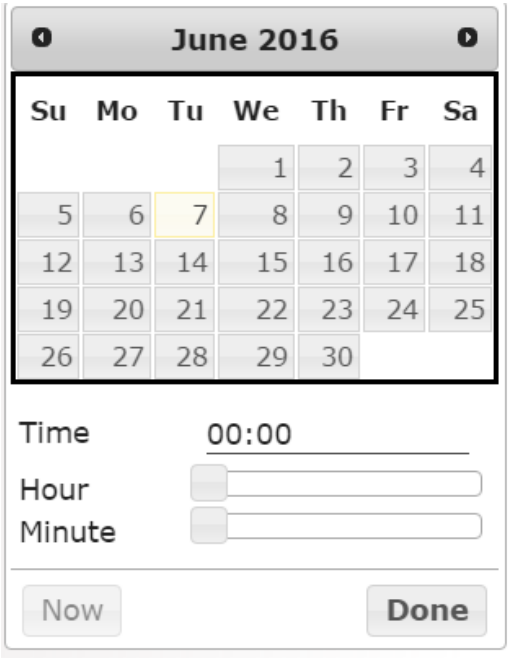
Het voorwaarde menu maakt gebruik van twee invoervelden. Deze invoervelden vormen een gevaar zoals beschreven in paragraaf 3.1.1. Daarom is ervoor gekozen om restricties te plaatsen op de mogelijke invoer. Dit kon makkelijk gedaan worden omdat de database voornamelijk numerieke data bevat. In het geval van numerieke data moet er een speciaal HTML-attribuut aan de invoervelden worden toegevoegd. Dit attribuut accepteert de invoer alleen op het moment dat er niks anders dan numerieke waardes zijn ingevoerd. Maar in het geval van een kolom met de



Figuur 8: Het voorwaarde menu.

naam “Date” mag de mogelijke invoer alleen bestaan uit een datumformaat. Om het invoerveld dynamisch aan te passen is jQuery gebruikt. Als de gebruiker verkeerde invoerwaardes geeft, zal er een popup te voorschijn komen die de gebruiker wijst op de fout. Naast de restrictie op de invoer is er ook een restrictie op het verzenden van een leeg invoerveld. Bij het kiezen van de ‘=’ operator kan invoerveld 2 leeggelaten worden. Dit betekent dat de gebruiker alle data van die kolom terug wil krijgen van de database. In het geval dat de operator niet gelijk is aan ‘=’ wordt de gebruiker verplicht om invoer te geven. Dit is nodig omdat de andere operatoren niet kunnen werken zonder invoer. Invoerveld 1 is in figuur 8 grijs gekleurd. Dit betekent dat het invoer veld op dit moment niet toegankelijk is. Invoerveld 1 word pas vrijgegeven als de operator ‘BETWEEN’ gekozen wordt.

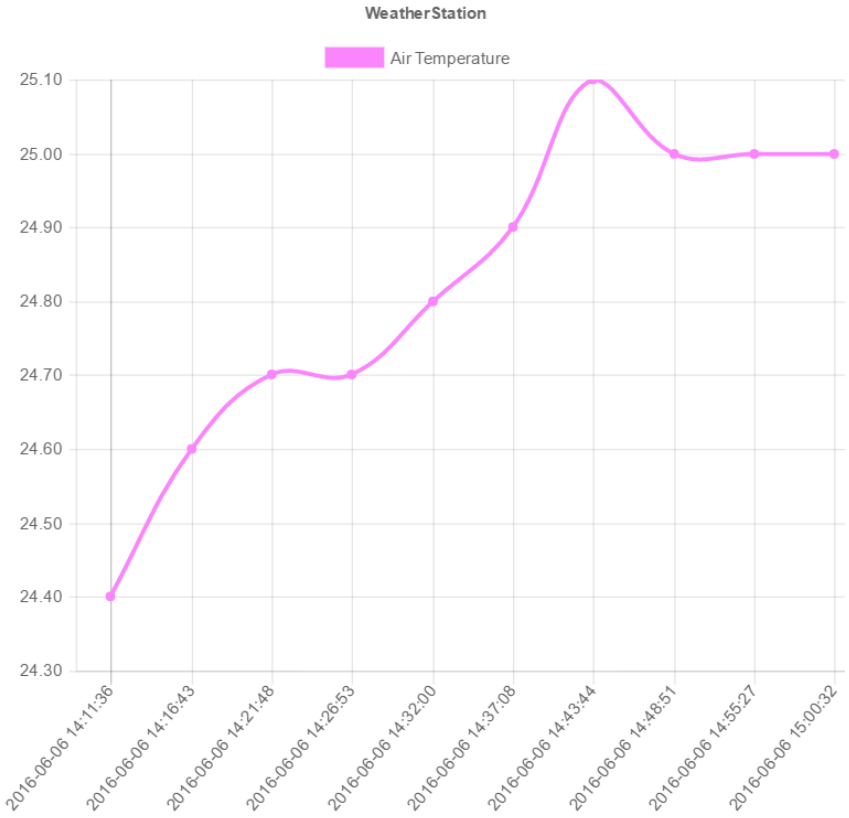
In het geval van een kolom met de naam “Date” is het invoeren van een datum en tijd zo makkelijk mogelijk gemaakt. De gebruiker krijgt na het klikken op het invoerveld van zo een kolom de kalender in figuur 9 te zien. Dit is een interactieve kalender waarop de gebruiker door een paar klik- en schuifbewegingen de datum en tijd kan kiezen. Deze datum zal automatisch ingevuld worden in het invoerveld. Voor het maken van de kalender is er gebruik gemaakt van de standaard jQuery kalender met een aanvulling om de tijd in te voeren [5].



Figuur 9: Kalender

3.2.3 Tabellen en grafieken

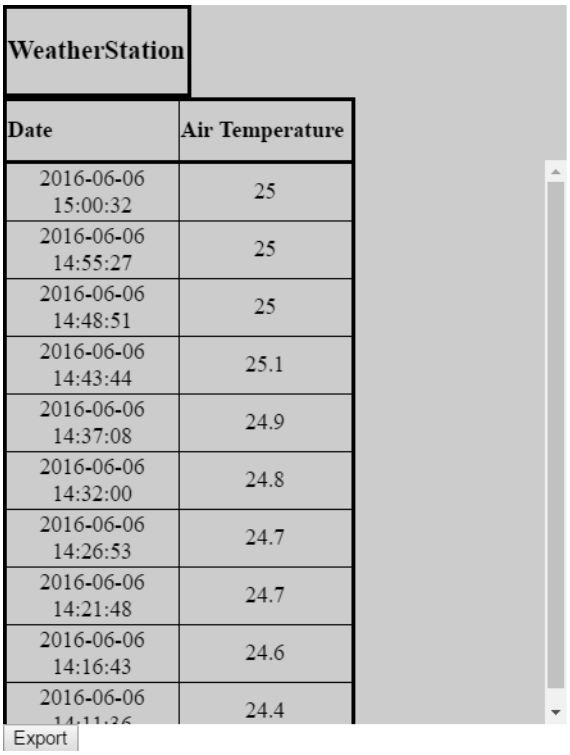
Zoals al te lezen was in de inleiding van dit hoofdstuk bevat de database vooral numerieke data. Daarom is ervoor gekozen om de data zichtbaar te maken in tabellen en grafieken, zoals te zien in figuur 10 en figuur 11. Per tabel is het mogelijk om een grafiek en/of een tabel met de gekozen data op te vragen [2.3]. Voor het maken van de grafieken is gebruik gemaakt van HTML5 canvas [2.4].Voor het tekenen op het canvas is de "Chart.js" bibliotheek gebruikt [3]. Het idee was om de grafieken volledig zelf te tekenen met behulp van alleen HTML5 canvas en JavaScript. Maar doordat dit veel tijd zou kosten is er voor de Chart.js library gekozen. Daarnaast beschikt deze library over een aantal zeer handige functies zoals het dynamisch veranderen van de grootte van de grafiek. Voor het maken van de tabel is er gebruik gemaakt van het HTML table-element. De tabellen en grafieken worden allemaal dynamisch gemaakt aan de hand van de gekozen waardes uit het checkbox menu. Daarnaast kan de gebruiker in het checkboxmenu aangeven of er een tabel of grafiek of beide gemaakt moet worden.



Figuur 10: Voorbeeld van een grafiek gemaakt met de chart.js bibliotheek.

Implementatie van de grafieken

Om het maken van de grafieken makkelijk te maken is er voor gekozen om een eigen functie te schrijven in jQuery die aan de hand van een array een grafiek kan maken. Daarnaast is het ook de bedoeling dat de andere groepen van dit project grafieken kunnen maken zonder de documentatie van Chart.js in te moeten duiken. Deze zelf gemaakte functie heeft de naam "makeChart" gekregen. Het vereenvoudigde overzicht van deze functie is te zien in figuur 12. In de gebruikersinvoer bevinden zich de tabelnamen, kolommen en hun waarden die opgehaald zijn door de functie getQuery. Aan de hand van de waarden in deze kolommen wordt er een grafiek gemaakt. Voor elke tabel wordt er een aparte grafiek gemaakt. Als eerst wordt de tabel naam boven aan de grafiek



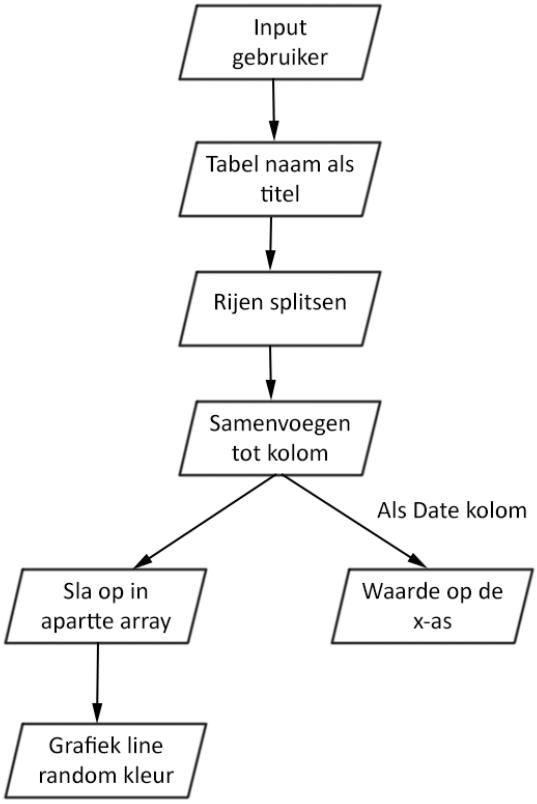
Figuur 11: Voorbeeld van een tabel.

geplaatst als titel. Hierdoor kan de gebruiker de grafieken makkelijk uit elkaar houden. Hierna moet de meegeven array gesplitst worden. Dit is nodig omdat in elke plaats in de array een rij van de tabel is opgeslagen. Deze rijen moeten opgesplitst worden en de juiste data van de juiste kolom moet allemaal bij elkaar gezet worden. Daarna word er onderscheid gemaakt tussen een date kolom en de andere kolommen. Omdat elke rij in een tabel een datum en tijd bevat is er voor gekozen om op de x-as altijd de datum en tijd te zetten. De overige kolommen zullen in aparte arrays opgeslagen worden. Elke lijn in de grafiek komt overeen met een kolom in de database en elke lijn krijgt ook een willekeurige kleur. Dit hele proces word per tabel herhaald.

Implementatie van de tabellen

De tabellen zijn gemaakt door middel van het HTML-element table. In dit element moeten de tabelnaam, kolomnamen en de waardes op de juiste plek in het element geplaatst worden. Ook de tabellen worden automatisch gegenereerd aan de hand van de keuzes van de gebruiker. Omdat de tabellen enkel op één plek gemaakt hoeven te worden, is er niet voor gekozen om er een losse functie van te maken. In tegenstelling tot de makeChart functie is het maken van de tabellen geprogrammeerd in PHP. De tabellen worden op de server gemaakt en daarna verzonden naar het dashboard die de tabel zichtbaar maakt. De tabellen worden gemaakt op de server omdat het makkelijker is om te werken met grote arrays in PHP dan in JavaScript.

Om de tabellen overzichtelijk te houden is er voor gekozen om de kolomnamen in de tabel altijd boven aan te laten zien. Zelfs bij het scrollen door de tabel moeten de kolomnamen zichtbaar blijven. Bij het scrollen in de verticale richting moeten de kolomnamen dus blijven staan. Maar bij het scrollen in de horizontale richting moeten de kolomnamen meebewegen zodat ze boven de juiste kolom blijven staan. Om dit te implementeren moest er handig gebruik gemaakt worden van CSS en JavaScript. Hierbij zorgt CSS ervoor dat de data in de kolommen scrollbaar is. In JavaScript word de positie van de scrollbar opgevraagd. Aan de hand van die positie wordt de positie van de kolomnamen aangepast.



Figuur 12: Het functie overzicht van de makeChart functie

De tabellen op de website zijn ook te downloaden in csv-formaat. Dit formaat wordt herkend door veel programma's zoals Microsoft Excel. Na het klikken op de export knop wordt er een bestand gedownload. Bij het openen van dit bestand word de tabel zichtbaar gemaakt in Excel. Dit is gedaan voor gebruikers die gebruik willen maken van de opgevraagde data. Voor het omzetten van de tabel naar csv is er gebruik gemaakt van de "jQuery table to csv plugin" [6]. Omdat de tabellen die gebruikt worden op het dashboard geen standaard tabelstructuur hebben, zijn de nodige aanpassingen aangebracht aan de plugin.

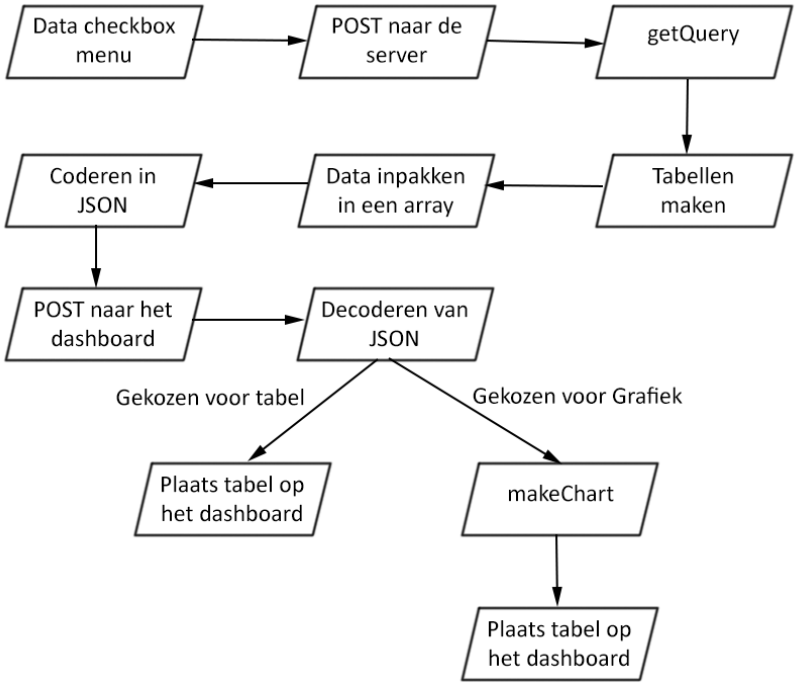
3.2.4 Communicatie met de server

Er zijn twee mogelijke methodes voor de communicatie met de server. De eerste methode is de GET methode. In dit geval wordt de data vanaf het dashboard verstuurd via de URL van het dashboard. Het voordeel is dat dit een simpele manier is om data te versturen. Het nadeel is dat alle data die verstuurd word in de URL te zien is. Dit is een rijke bron van informatie voor kwaadwillenden. Daarnaast kan de URL maar een beperkte hoeveelheid data bevatten.

De tweede methode maakt gebruik van de POST methode. Bij de POST methode wordt de data niet via de URL verstuurd maar via een HTTP bericht. Een aantal voordelen hiervan zijn dat de data niet zichtbaar is in de URL en er zit geen limiet op de hoeveelheid data die verstuurd kan worden. Hierdoor is de data beter beschermd. Het nadeel is dat het moeilijker is om met de POST methode te werken. Omdat het checkboxmenu erg veel data heeft om te verzenden en deze data beschermt moet worden, is er voor gekozen om de POST methode te gebruiken. Naast het verzenden van de data naar de server, wordt de data verzonden vanaf de server ook via een POST gedaan.

Implementatie
Zoals te lezen was in paragraaf 3.2.3 worden de tabellen op de server gemaakt. Deze tabellen worden gemaakt aan de hand van de data verstuurd vanaf het dashboard in een POST. In figuur 13 is een vereenvoudigde weergave van de communicatie te zien. De data die verstuurd wordt naar de server wordt gekozen aan de hand van het checkboxmenu. Deze data wordt verzonden door op de 'get data' knop te drukken. Bij aankomst op de server word aan de hand van de data van het checkbox menu, de database aangesproken via de getQuery functie. De data die teruggegeven wordt van de database wordt opgeslagen in een array en er worden tabellen gemaakt. De tabellen worden aan het eind van de array geplaatst. Dit is zo gedaan zodat de data van de server in één POST verzonden kan worden. Het zou ook in meerdere POST's kunnen maar kijkend naar de veiligheid van de data is dit niet wenselijk. Voor het versturen wordt de volledige array gecodeerd naar JSON. Dit staat voor JavaScript Object Notatie, en wordt voornamelijk gebruikt door de taal JavaScript. Deze notatie heeft een flexibele notatie voor alle soorten datastructuren, en daarom is deze gebruikt in plaats van de standaard manier van POST data versturen.

Aan de dashboard kant word er gebruik gemaakt van AJAX om de data van de server te ontvangen en te verwerken. AJAX is een manier om een website aan te passen zonder de pagina te herladen [13]. jQuery bevat de nodige functies om gebruik te maken van AJAX. Na het ontvangen van de data moet de data eerst gedecodeerd worden van JSON naar objecten en arrays. Nu kunnen de tabellen direct aan de webpagina toegevoegd worden als de gebruiker ervoor gekozen heeft om tabellen te laten zien. Op het moment dat de gebruiker voor grafieken gekozen heeft moeten deze nog gemaakt worden. Dit word gedaan aan de hand van de array opgestuurd vanaf de server. Deze array word meegeven aan de functie makeChart die er een grafiek van maakt.



Figuur 13: Vereenvoudigde weergave van de communicatie tussen de server en het dashboard.

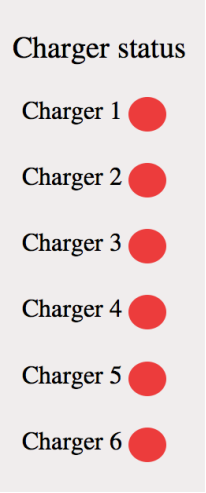
3.2.5 Charger menu

Om aan eis [2.1] te voldoen, is er aan de rechterkant van het scherm een menu gemaakt met zes knoppen. Dit is het charger menu te zien in figuur 14. Deze 6 knoppen geven door middel van hun kleur de status van de opladers weer. Groen betekent dat de oplader aanstaat en dat er dus

een fiets aan het opladen is. Rood betekent dat de oplader uitstaat. De status van de opladers is opgeslagen in de database. Bij het laden van het dashboard worden de statussen van de laders opgehaald met behulp van de getQuery functie. Afhankelijk van de waarde in de database krijgt de status knop een bepaalde kleur.

Door op één van de knoppen te klikken kan de status van een oplader veranderd worden. Als de vorige status aan was wordt de knop na erop te klikken dus rood en gaat de oplader uit. Andersom als de status uit was word de knop na erop te klikken groen en gaat de oplader aan. Het veranderen van de status in de database word gedaan door de updateQuery functie te gebruiken.

Als de ODROID aangeeft dat de lader uit wordt gezet, verandert de status in de database. Om de pagina actueel te houden, wordt elke anderhalve seconde de huidige status opgehaald. Om te verhinderen dat de website zich elke anderhalve seconde herlaadt, wat ervoor zou zorgen dat selecties verloren raken, wordt ook hiervoor AJAX gebruikt. Er wordt een array ontvangen in JSON formaat met zes enen of nullen. Deze array wordt gedecodeerd en vervolgens wordt voor elke lader waar een '0' voor ontvangen is een rode knop gemaakt en voor de laders met een '1' een groene knop.



Figuur 14: Het opladermenu

3.2.6 Inlogsysteem

Voor het beschermen van het systeem tegen kwaadwillenden zijn al veel maatregelen genomen. Zo is het alleen mogelijk om de database aan te spreken door gebruik te maken van de zelfgemaakte functies uit paragraaf 3.1. Daarnaast word de invoer mogelijkheid van de gebruiker beperkt zoals te lezen in paragraaf 3.2.2. Maar de belangrijkste manier van beveiligen is het gebruik van een inlogscherm. Voordat de gebruiker op het dashboard kan komen moet er ingelogd worden via het app login scherm. Dit is hetzelfde scherm als waar de gebruikers van de oplaadpaal inloggen. De administrator kan een gebruiker de rechten geven om in te kunnen loggen op het dashboard. Dit word gedaan door in de database het type van het account te veranderen.

3.3 Communicatie tussen ODROID en server

3.3.1 Basis

In sectie 1.2 is genoemd dat er al code beschikbaar was. Deze code was echter niet beschermt tegen SQL injecties beschreven in sectie 3.1.1. Ook werd de meest onveilige manier gebruikt om data te verzenden, door alle data in de url van de ontvangende webpagina te zetten. Om deze redenen is besloten om deze code niet te gebruiken en nieuwe te maken.

De relevante eisen staan hier nog een keer genoemd.

[1.1] De code op de server moet in PHP en MySQL geschreven worden;

[3.1] De ODROID moet meetdata kunnen opsturen naar de server;

[3.2] Als de connectie met het internet wegvalt, moet deze meedata gebufferd worden om later alsnop op te sturen;

[3.3] Als de buffer vol is, dan vervallen de eerste waardes om plaats te maken voor nieuwe;

[3.4] De ODROID moet aan de server kunnen laten weten wanneer een fiets losgekoppeld is, zodat de reserveerpagina deze plekken weer kan aanbieden;

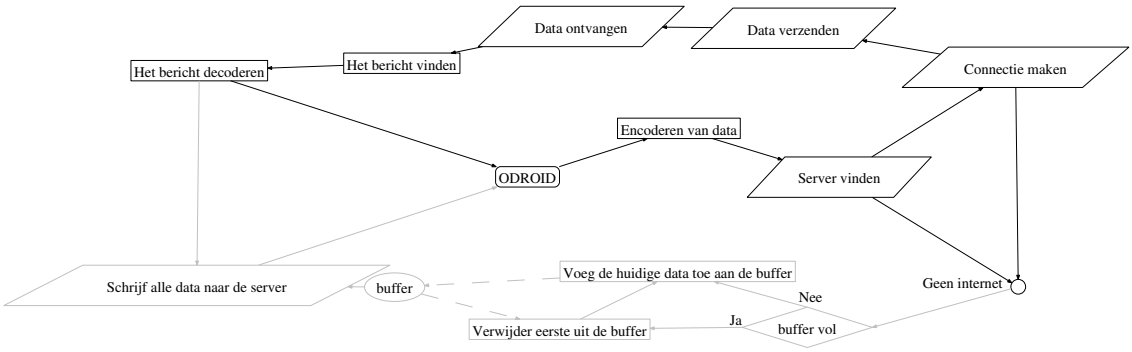
[3.5] De ODROID moet de huidige status van de laders kunnen opvragen, zodat de correcte laders stroom kunnen krijgen;

[3.6] De code op de ODROID mag geen geheugenlekken bevatten en moet altijd blijven werken.

[3.7] De code op de ODROID moet in C geschreven worden.

De communicatie tussen de ODROID en de server gebeurt allemaal via de notatie genaamd “JSON”. Aan de server kant is het ontsleutelen eenvoudig doordat PHP de nodige functies bevat die JSON direct omzetten. Aan de ODROID zijde zijn speciale functies geschreven die de data encoderen en decoderen.

In figuur 15 is de basis te zien van het programma op de ODROID. De ODROID leest data uit de meetsystemen en moet deze versturen. Voor dat de data verstuurd kan worden, word de data eerst geëncodeerd naar JSON formaat. Er worden nog wat extra gegevens toegevoegd aan de JSON, zoals het type data (komt het van de Victron, of toch van het weerstation), en het huidige tijdstip. Daarna wordt de internet verbinding geopend, en wordt het IP adres van de server opgezocht. Als deze gevonden is, wordt met dit adres geprobeerd om een verbinding op te zetten met de server. Als dat lukt wordt het bericht dat de data bevat, samen met de standaard gegevens die elk internet bericht moet hebben opgestuurd naar de server.



Figuur 15: De basis uitlijn van de communicatie tussen de ODROID en de server

De server verwerkt deze data, en stuurt mogelijk weer data terug in de vorm van JSON. Deze data wordt ingelezen, en met behulp van de standaard informatie die de server terugstuurt, de

"header", wordt de lengte van het bericht bepaald. Dit is nodig zodat bepaald kan worden hoelang de string moet zijn waarin het eigenlijke bericht van de server staat. Ook wordt in de code het einde van de header gevonden. De resterende data wordt even apart gezet, om gedecodeerd te worden zodat de ODROID deze data weer kan gebruiken.

Voor de connectie vanaf de ODROID is voor de low-level methode van sockets gekozen. Deze zijn op alle UNIX systemen hetzelfde gedefinieerd, dus geven weinig problemen als het op een andere (Linux of Mac) computer moet worden getest. Ook zijn de andere beschikbare methodes trager, want ze moeten zelf deze sockets gebruiken en ze hebben zelf code daaromheen geschreven. Hierdoor worden beide delen van de code aangeroepen en is de computer langer bezig met uitvoeren. Ook zijn deze methoden niet standaard op alle computers aanwezig, dus ze moeten eerst geïnstalleerd worden.

Er zijn twee methodes om de data op te sturen voor de verschillende tabellen in de database. De eerste manier is dat de verschillende data naar verschillende webadressen wordt verstuurd, waar PHP-scripts de data in de database zetten. De tweede methode is dat alle data naar hetzelfde webadres gaat, waarbij wat extra informatie wordt opgestuurd om de server te laten weten waar de data vandaan komt. Omdat de code aan de serverzijde voor alle soorten data op een paar variabelen na hetzelfde is, is gekozen om maar één webpagina te maken en een getal op te sturen als "code" zodat de server weet welke data het is.

Implementatie

Allereerst is informatie over sockets opgezocht. Dit leverde de officiele documentatie [14] op en een website met een voorbeeld in C [15]. Na het testen van deze code is geprobeerd om de code aan de gebruikerszijde te laten verbinden met de server. Daarna werd gevonden welke extra informatie, de zogeheten "headers", mee opgestuurd moesten worden. Dan werd getest of data in JSON-formaat kon worden verzonden en ontvangen. Toen dat werkte, is begonnen met het schrijven van de functies, gebaseerd op het schema in figuur 15.

In het bestand waarin alle functies staan die de data van de ODROID ontvangen en ze opstuurt, is een functie gemaakt die alle bovenstaande functies aanroept. Dit is ook de functie die de type data meestuurt in de JSON.

Hierna zijn de verschillende functies een aantal keer geoptimaliseerd en zijn verschillende geheugenlekken eruitgehaald.

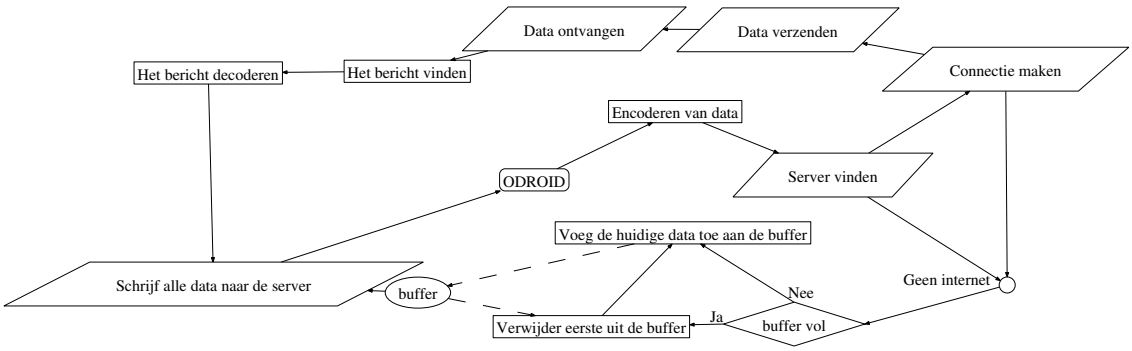
3.3.2 Bufferen

Als er geen internet is, kan de ODROID geen waardes opsturen naar de server. Om ervoor te zorgen dat de server zo min mogelijk waardes verliest, is er een mechanisme ingebouwd die de berichten van de ODROID buffert. Als het internet het een tijdje later opeens wel weer doet, wordt de data alsnog opgestuurd naar de server.

Als de ODROID erg lang geen connectie kan maken, zouden heel veel berichten opgeslagen moeten worden. Maar de ODROID heeft een eindig geheugen. Daarom is er voor gekozen om slechts 200 berichten te bufferen. Als, nadat de ODROID ruim acht uur geen internet heeft gehad, het 201e bericht binnenkomt, wordt de eerste weer verwijderd.

In figuur 16 is een basisschema te zien van de layout. Op het moment dat de data ontvangen is en gedecodeerd, loopt de ODROID door de buffer heen en stuurt de waardes in de buffer alsnog op. Als er weer iets met het internet aan de hand is, wordt het uitlezen gestopt en worden de laatste berichten tot een ander moment bewaard.

Dit leverde een probleem op. De datum en tijd werden namelijk aan de serverzijde bepaald zodat in de database de goede datum en tijd stonden. Maar als de data eerst moet bufferen, en er tien verschillende database punten voor dezelfde tabel achter elkaar snel verstuurd worden, komen ze allemaal in de database met het tijdstip van ontvangst. Dat gaat er vreemd uitzien in de grafieken. Omdat op te lossen bepaalt de ODROID de huidige datum en tijd, en stuurt dat samen met alle andere informatie mee naar de server. Dit zorgt ervoor dat het tijdstip ook meegebufferd wordt als het internet het niet doet. Hierdoor klopt de datum en de tijd in de database.



Figuur 16: De basis uitlijn van de communicatie met de buffer toegevoegd

Implementatie
Nadat het ontwerp, beschreven in paragraaf 3.3.1, afgerond was, is begonnen met een functie die een string in een buffer kan plaatsen en een andere functie die de buffer leegt en zoveel mogelijk van deze strings opstuurt naar de server. Nadat deze functies goed werkte is de allesomvattende functie aangepast zodat deze de data ging bufferen als er geen internet is. Een uitzondering is gemaakt: Als de server om de status van de laders vraagt, en er is geen internet, dan wordt niets teruggegeven. Anders had de server misschien een uur later de aanvraag pas beantwoord en was de data niet actueel meer.

Ook deze functies zijn een aantal keer veranderd in de loop van de tijd om lekken op te lossen en om de code te optimaliseren.

3.3.3 Revisie

In de bovenstaande code werd gebruik gemaakt van zogeheten “globale variabelen”. Dat betekent dat de variabele overal in de code beschikbaar is. Dit leverde al ergens een probleem waar een van deze variabele overschreven werd wat weer leidde tot geheugenlekken. Om dit soort problemen in de toekomst te voorkomen is besloten om de code deels te herschrijven. Er is besloten om een data structuur te maken met alle globale variabelen erin. Elke keer dat er data opgestuurd wordt, wordt er een nieuwe structuur aangemaakt en wordt alles gedaan met deze structuren. Als al het werk gedaan is wordt het structuur weer opgeruimd om geheugen te besparen. Slechts één variabele moest globaal blijven: de buffer voor als het internet het niet doet. Anders wordt de data gebufferd in een data structuur als het internet het niet doet, vervolgens wordt het structuur opgeruimd en is de data verloren. Er is in C geen betere manier om dit op te lossen.

3.3.4 Serverzijde

Aan de serverzijde wordt allereerst de data van de invoer gelezen. Daarna wordt deze data gedecodeerd vanuit JSON. Het type data wordt ontrafeld en daaruit wordt beslist wat er gedaan moet worden:

- Bij het type 100 (test) wordt een reeks waardes aangemaakt waar de ingekomen waardes bijgevoegd worden. Deze wordt gebruikt voor tests van de C-code, voornamelijk voor het encoderen en het decoderen van JSON.
- Bij het type 10 (verkrijg de status van de laders) wordt de huidige status van alle laders uit de database gelezen. Deze zes waardes worden teruggestuurd.
- Bij de types 0 tot en met 5 wordt de data in de corresponderende tabel gezet. Het meegezonden tijdstip (in seconden sinds 1 januari 1970 00:00) wordt vertaald in een leesbaarder formaat, of als er om een of andere reden geen tijdstip meegezonden is, bepaalt de server deze. In dit geval stuurt de server niets terug, omdat de ODROID deze waardes negeert.

- Bij het type 20 worden de laders in de database uitgezet die door de ODROID vrij zijn gegeven. De ODROID stuurt een '1' voor alle laders die verlaten zijn in de laatste paar seconden. Daarmee wordt de database geüpdatet.

Voor deze laatste functionaliteit is, om de beveiliging te waarborgen, gebruikgemaakt van de functie "insertQuery", behandeld in sectie 3.1.1. De code om de status uit te lezen gebruikt de "getQuery" functie niet, want in de tijd dat die code geschreven is, was de getQuery functie nog niet veilig. In plaats daarvan worden de functies van MySQLi direct aangeroepen.

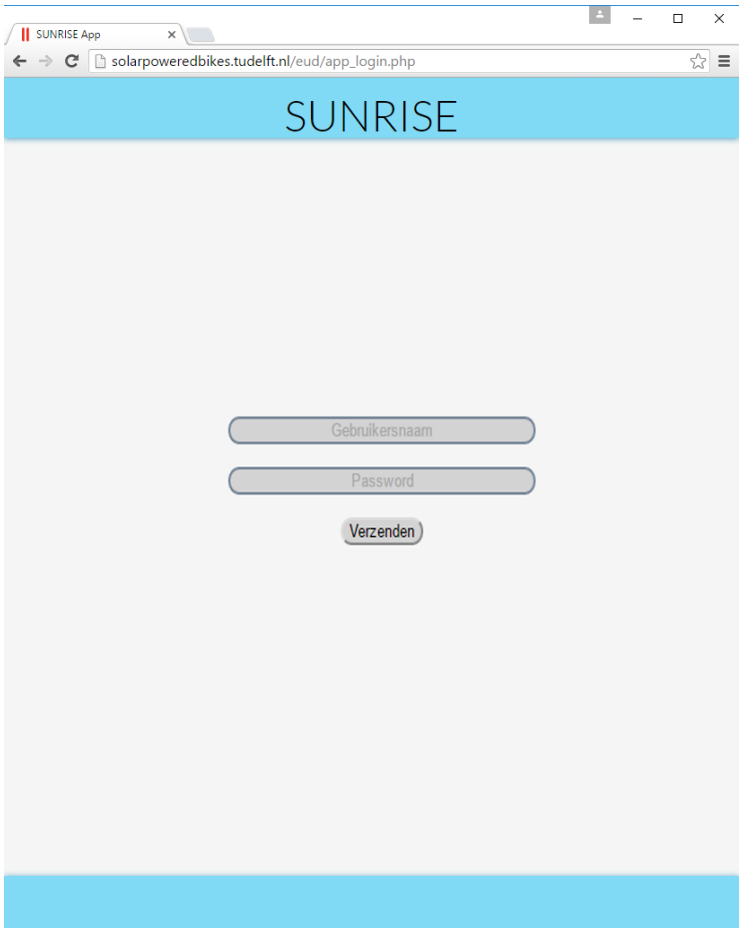
Implementatie
Allereerst was een klein PHP-script geschreven om te kijken of dat er verbinding gemaakt kon worden. Toen deze webpagina correct terugkwam aan de C-zijde, werd er geprobeerd om JSON op te sturen. Aan de PHP-zijde bleek dat het lezen van de data handmatig moet gebeuren. Met de normale POST-formaat is er een speciale variabele waarin de data beschikbaar is. Nu bleek dat er van de "input" gelezen moest worden [16]. Daarna moest de data gedecodeerd worden via de ingebouwde functies, voordat de data beschikbaar is. Vervolgens stuurde de PHP een andere array terug, met de ontvangen data er in, geëncodeerd in JSON.

Toen dit werkte, werd begonnen met de uiteindelijke code. Het getal wat de type data aangeeft wordt gebruikt om te bepalen welke code uitgevoerd moet worden. Allereerst werd de code om de laders uit te lezen geschreven. Daarna werd begonnen met de code die de meetdata in de database zette. Daarin wordt eerst, per type data, de tabelnaam bepaald en de kolomnamen van de tabel in de juiste volgorde neergezet. Daarna wordt de datum geformatteerd zodat deze op de goede manier in de database komt te staan. Dan wordt de insertQuery functie aangeroepen. Er wordt in dit geval niets teruggegeven.

Als laatste is de code voor het bijwerken van de opladers geschreven. De data komt binnen als array van zes waardes, waarbij alle waardes een '1' of een '0' zijn. De '1' geeft aan dat de lader vrijgegeven is door de gebruiker. De code itereert door de array, en als er een '1' staat in het betreffende element, wordt de index van dat element gebruikt om de betreffende rij in de database te updaten met de "updateQuery" functie, beschreven in sectie 3.1.1.

4 Resultaten

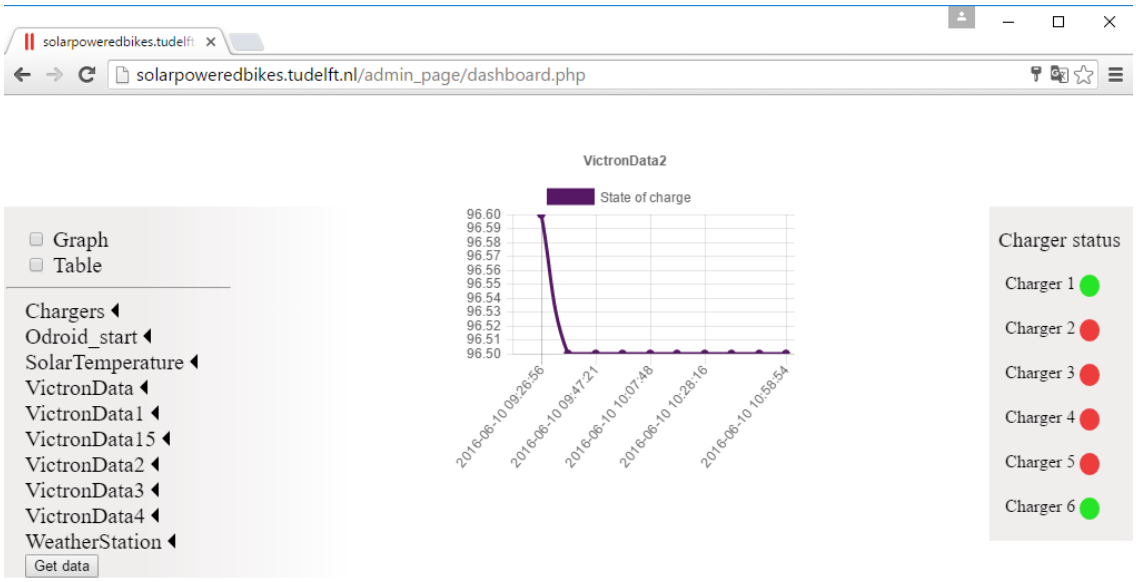
Het systeem voldoet nu aan alle opgegeven eisen. Het dashboard staat online en alle data is op te vragen van de database. Voordat de gebruiker bij het dashboard kan komen moet er ingelogd worden via het inlogscherm te zien in figuur 17. Hier wordt gevraagd om een gebruikersnaam en wachtwoord. Voor een account waarmee men kan inloggen op het dashboard moet er een speciaal account aangevraagd worden bij de administrator. Als er wordt ingelogd met een normaal account zal de gebruiker doorgestuurd worden naar de app.



Figuur 17: De login pagina.

Na het inloggen komt het dashboard pagina zoals in figuur 18 tevoorschijn. De actuele informatie over de batterijspanning en de opladers is direct op de pagina te zien. Er kan ingelogd worden met de volgende gegevens:
Gebruikersnaam: testadministrator
Wachtwoord: BAP

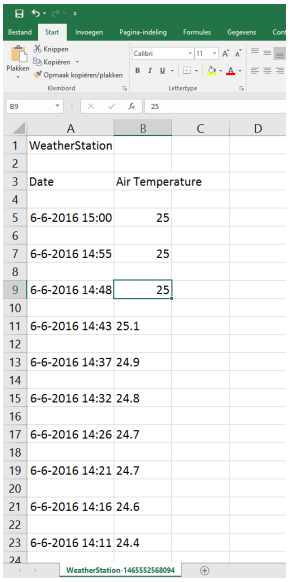
Door middel van het keuze menu aan de rechterkant van het scherm kunnen er meerdere tabellen en grafieken gemaakt worden aan de hand van alle data in de database. Eerst moet er gekozen worden tussen een tabel en of een grafiek. Daarna kan het menu met chechkboxes van een bepaalde tabel geopend worden. Dit wordt gedaan door op het zwarte driehoekje te klikken. In het checkbox menu zijn alle kolommen van een bepaalde tabel te vinden. Hierbinnen zijn er weer zwarte driehoekjes te vinden. Na het klikken op een driehoekje wordt het voorwaarde menu geopend. Hier heeft de gebruiker de mogelijkheid om voorwaardes mee te geven. Als de gebruiker



Figuur 18: Het dashboard direct na het inloggen.

klaar is met het kiezen van de juiste data moet er op de get data knop geklikt worden. Nu verschijnen de tabellen en of grafieken direct op het dashboard. Deze tabellen en grafieken zien eruit zoals in figuur 10 en 11.

Een tabel kan gedownload worden door op de export knop te drukken. Na het downloaden kan het bestand geopend worden in een progamma zoals Microsoft Excel. De tabel komt eruit te zien zoals in figuur 19.



	A	B	C	D
1	WeatherStation			
2				
3	Date	Air Temperature		
4				
5	6-6-2016 15:00	25		
6				
7	6-6-2016 14:55	25		
8				
9	6-6-2016 14:48	25		
10				
11	6-6-2016 14:43	25.1		
12				
13	6-6-2016 14:37	24.9		
14				
15	6-6-2016 14:32	24.8		
16				
17	6-6-2016 14:26	24.7		
18				
19	6-6-2016 14:21	24.7		
20				
21	6-6-2016 14:16	24.6		
22				
23	6-6-2016 14:11	24.4		

Figuur 19: De tabel geëxporteerd naar Excel.

4.1 Communicatie tussen ODROID en server

In deze sectie worden de tests die gedaan zijn voor de communicatie tussen de ODROID en de server nader beschreven.

4.1.1 Verbinding tussen ODROID en server

Allereerst werd geprobeerd om de voorbeeldcode beschreven in [15] te gebruiken om connectie te maken met de server. De resultaten hiervan staan in lijst A.1.1 in appendix A.1

Vervolgens is de code verbeterd zodat deze alle informatie bevat wat teruggestuurd wordt. Het teruggestuurde bericht was namelijk niet volledig. Het had moeten eindigen met '</html>'. Ook werd gekeken welke informatie meegestuurd moest worden. Hiermee kwam de tweede test. Hiervan staan de resultaten in lijst A.1.2

4.1.2 JSON

Nadat de connectie op orde was zijn de tests gedaan om de JSON te encoderen en decoderen. Allereerst was een test gedaan die aan de serverzijde de JSON ontving en decodeerde, zie lijst A.2.1 in appendix A.2

Daarna werd getest of de server ook JSON terug kan sturen en of dit geëncodeerd en gedecodeerd kon worden. De server verzond dezelfde waardes die het binnenkreeg, met een aantal waardes toegevoegd. De resultaten staan in A.2.2

4.1.3 MySQL

Toen de JSON tests gedaan waren werd begonnen met de MySQL implementatie. Daarvoor werden meerdere tests gedaan. Hieronder staat de test die gebruikt is om de status van de laders te verkrijgen van de server.

```
1 $ ./test
2 1
3 0
4 1
5 0
6 1
7 1
```

4.1.4 Buffer

Er is één programma geschreven om de buffer te testen. Dit programma laat de gebruiker zijn internet uitzetten en probeert dan iets naar de server op te sturen. Vervolgens moet het internet weer aangezet worden en dan wordt er weer iets naar de server opgestuurd. Als alles werkt, worden twee berichten tegelijkertijd ontvangen. De tijdstippen die in de database verschijnen, moeten dezelfde tijdstippen zijn als de tijdstippen van de pogingen. Bijvoorbeeld, als de poging terwijl het internet uitstond om 12:01:05 was, en de poging terwijl het internet aanstaat om 12:01:20, dan moeten deze beide tijdstippen in de database te zien zijn na de test.

```
1 $ ./test
2 Zet internet uit, toets enter om verder te gaan
3
4 ERROR, cannot find host: solarpoweredbikes.tudelft.nl
5 Zet internet nu weer aan, toets enter om verder te gaan
```

←T→	Date 1	ID	Grid L1 - Power	Grid L2 - Power	Grid L3 - Power	Grid L1 - Energy from net	Grid L2 - Energy from net	Grid L3 - Energy from net	Grid L1 - Energy to net	Grid L2 - Energy to net	Grid L3 - Energy to net
Wijzigen Kopiëren Verwijderen	2016-06-08 15:16:42	82	21	29	8	2	7	38	22	36	3
Wijzigen Kopiëren Verwijderen	2016-06-08 15:16:36	83	21	29	8	2	7	38	22	36	3

Figuur 20: De database rijen die bij test 4.1.4 horen. De data in de kolommen zijn willekeurig gekozen getallen, slechts de datum en tijd en de ID is actueel.

4.1.5 Valgrind

Alle tests die hierboven genoemd worden, zijn allemaal minstens twee keer gedaan. Toen de normale test goed ging, werd nog een keer getest met het programma “Valgrind” [17]. Dit programma bevat verschillende gereedschappen waarmee verschillende fouten gevonden kunnen worden, zoals multitask fouten, fouten met het geheugen van c-code. Daarnaast bevat Valgrind ook simalatiesoftware om statistieken te bepalen. Voor de tests werd het gereedschap “Memcheck” gebruikt [18]. Deze kan geheugenlekken en andere soorten geheugenfouten vinden. Door deze tests kon gevonden worden waarom programma’s vreemd gedrag vertonen en soms zelfs stoppen.

In appendix A.3 staat een voorbeeld van hoe een test er uit zou zien met Valgrind. Valgrind begint met een copyright te printen, en welk gereedschap gebruikt wordt. Daarna voert het de test uit die meegegeven is, en waar nodig print het waarschuwingen tussendoor. Op het einde geeft Valgrind een samenvatting van het geheugengebruik en hoeveel geheugen er weggelekt is.

5 Discussie

Het gehele ontwerp en implementatie proces is goed verlopen. De data van de database is beschikbaar gemaakt voor mensen die geen kennis hebben van SQL. In plaats van SQL hoeft er alleen gebruik gemaakt te worden van het checkboxmenu wat heel simpel te gebruiken is. Veel websites gebruiken een soortgelijk menu. Maar bij het aanpassen van de structuur van de database, moet de code van de webpagina aangepast worden. Zonder het aanpassen van de webpagina code kan het zo zijn, dat het gehele menu niet meer toegankelijk is. Dit probleem is in het geval van het dashboard echter niet van toepassing. Doordat het menu, telkens als de webpagina herladen wordt, volledig opnieuw gemaakt wordt. Dit gebeurt zonder dat de gebruiker er ook maar iets van merkt.

Het datum formaat in de database was anders dan het formaat gemaakt door de jQuery kalender. Om dit op te lossen moest de datum vanuit de kalender omgezet worden. Dit is gedaan door gebruik te maken van de library “Moment.js” [4]. Moment.js bied de mogelijkheid om een datum formaat te converteren naar elk gewenst formaat. In plaats van deze oplossing had de database aangepast kunnen worden. Alleen omdat er veel meetgegevens verloren zouden gaan is er voor de eerst genoemde oplossing gekozen.

De eis dat de ODROID voor altijd moet kunnen blijven werken is lastig te controleren. Er zijn geen geheugenlekken in de code gevonden tijdens het testen, maar het is niet zeker dat dit garandeert of de ODROID oneindig lang kan blijven werken. Als er iets met het internet gebeurt waardoor er vreemde data terugkomt is er een kleine kans dat de code niet meer werkt en de ODROID daardoor stopt met werken.

6 Conclusie

Het project is geslaagd. De gebruiker kan bij aankomst bij de fietsenstalling een oplaadpunt reserveren. Hiervoor kan de gebruiker inloggen op de app en een oplaadpunt kiezen. Groep Server zorgt voor de communicatie tussen de server en de database en tussen de server en de ODROID. Ook moest er een dashboardpagina gemaakt worden waarop alle data uit de database op te vragen is. Het systeem voldoet aan alle eisen. De communicatie met de database verloopt op basis van de PHP MySQLi bibliotheek. De database is een groot doelwit voor kwaadwilleden en moet dus beschermd worden. Alle communicatie met de database is beveiligd door gebruik te maken van de zelf gemaakte functies: insertQuery, updateQuery en getQuery. Door deze functies is het voor een kwaadwillende niet mogelijk om de database query's uit te laten voeren die niet toegestaan zijn.

De ODROID kan communiceren met de server. Vanuit het oplaadstation verzendt de ODROID alle data naar de server. De server zorgt er verder voor dat de data in de database te recht komt. De ODROID vraagt de status van de laders op. Aan de hand van deze status word een lader aan- of uitgezet. Op het moment dat een gebruiker zijn fiets ontkoppelt geeft de ODROID dit door aan de server.

De beheerder kan de dashboardpagina bekijken door in te loggen. Daar kan deze de data uit alle tabellen in de database opvragen. Het checkboxmenu zorgt voor een zeer gebruiksvriendelijke manier om de gewenste data te kiezen. Alle data kan zichtbaar gemaakt worden in tabellen en of grafieken. Op de pagina is ook de status van de opladers af te lezen. Daarnaast is het zelfs mogelijk om de opladers via het dashboard aan en uit te zetten.

6.1 Aanbeveling voor de toekomst

Het systeem werkt goed, maar er is altijd ruimte voor verbetering. De structuur van de HTML zou na de kennis opgedaan in dit project minder complex kunnen. Dit heeft verder geen invloed op de werking van het menu. Het zou het alleen makkelijker maken om CSS toe te passen op het checkboxmenu. Hierdoor kan het uiterlijk van de pagina verbeterd worden.

Na het gebruik van HTML5 canvas zijn er veel ideeën bijgekomen voor het tonen van data. Naast de tabellen en grafieken zou het mogelijk zijn geweest om de data op meer illustratieve manieren te laten zien. Bijvoorbeeld het maken van een geanimeerde batterij die aangeeft hoeveel lading er in de batterijen zit. Dit is niet geïmplementeerd, aangezien de hoeveelheid tijd voor dit project maar beperkt is.

Op de x-as van de grafiek is altijd de datum en tijd te zien. Alleen kloppen de stappen tussen de tijden niet altijd. Dit komt doordat de data niet altijd op het zelfde tijdsinterval ontvangen wordt. Dit zou opgelost kunnen worden door een eigen versie van de chart.js bibliotheek te schrijven. Waarin je het interval tussen de punten makkelijk kunt aanpassen.

In de toekomst zou het handig zijn om op het dashboard ook te kunnen zien of er werkelijk een fiets staat geparkeerd. Op dit moment kan iemand een oplaadpunt bezet houden zonder op te laden. Hierdoor kan het zo zijn dat de gebruiker een plaats reserveert die al bezet is door een fiets die niet ingeplugd is.

Er is een probleem waarbij laders onbruikbaar kunnen worden. Als iemand zijn e-bike ontkoppeld, terwijl er een internetstoring is, wordt het bericht dat de lader vrijgegeven is gebufferd. Als het internet er daarna meer dan acht uur uit ligt, wordt deze informatie over de lader verwijderd uit de buffer, om plaats te maken voor nieuwe berichten. Dus de server ontvangt nooit meer het bericht dat de lader vrij is, en deze lader kan niet meer gereserveerd worden. Deze fout zou opgelost kunnen worden door een andere buffer te maken voor het updaten van de laders dan voor het bufferen van de meetdata.

Het beveiligen van de server is voornamelijk gedaan door de invoer van de gebruiker te controlleren en te beperken. Het idee was om hiernaast ook nog eens gebruik te maken van HTTPS [19]. HTTPS is een manier om communicatie over het internet te versleutelen. Voor het gebruik van HTTPS moet er een aanvraag ingediend worden bij de TU Delft. Als de HTTPS geregeld is, moet de C-code aangepast worden zodat het bericht versleuteld wordt, bijvoorbeeld via de bibliotheek "OpenSSL" [20].

Referenties

[1] Multiple authors. (2016) phpMyAdmin. [Online]. Available: https://www.phpmyadmin.net/

[2] ——. (2016) jQuery API documentation. [Online]. Available: https://api.jquery.com/

[3] ——. (2016) Chart.js. [Online]. Available: http://www.chartjs.org/

[4] ——. (2016) Moment.js. [Online]. Available: http://momentjs.com/

[5] T. Richardson. (2016) Adding a Timepicker to jQuery UI Datepicker. [Online]. Available: http://trentrichardson.com/examples/timepicker/

[6] C. Thomas. (2015) jQuery.tabletoCSV. [Online]. Available: https://github.com/cyriac/jquery.tabletoCSV

[7] Multiple authors. (2016) SQL Injection. [Online]. Available: http://www.acunetix.com/websitesecurity/sql-injection/

[8] ——. (2016) MySQL :: MySQL 5.7 Reference Manual :: 14 SQL Statement Syntax. [Online]. Available: http://dev.mysql.com/doc/refman/5.7/en/sql-syntax.html/

[9] ——. (2016) PHP: MySQLi - Manual. [Online]. Available: http://php.net/manual/en/book.mysqli.php

[10] ——. (2016) Mediamarkt. [Online]. Available: https://www.mediamarkt.nl/

[11] ——. (2016) Coolblue. [Online]. Available: https://www.coolblue.nl//

[12] ——. (2016) Fooplot. [Online]. Available: http://fooplot.com/

[13] ——. (2016) jQuery.ajax() | jQuery API Documentation. [Online]. Available: http://api.jquery.com/jquery.ajax/

[14] ——. (2008) The GNU C library. [Online]. Available: http://www.gnu.org/software/libc/manual/html_mono/libc.html#Sockets

[15] ——. (2013) Sockets Tutorial. [Online]. Available: http://www.linuxhowtos.org/C_C++/socket.htm

[16] ——. (2016) PHP: php:// - Manual. [Online]. Available: http://php.net/manual/en/wrappers.php.php

[17] ——. (2015) Valgrind. [Online]. Available: valgrind.org

[18] ——. (2015) 4. Memcheck: a memory error detector. [Online]. Available: http://valgrind.org/docs/manual/mc-manual.html

[19] R. Abela. (2014) WordPress SSL and HTTPS Explained. [Online]. Available: https://www.wpwhitesecurity.com/wordpress-tips-webmasters/wordpress-blog-website-ssl-https-explained/

[20] Multiple authors. (2015) OpenSSL. [Online]. Available: https://www.openssl.org/

Appendix

A Testresultaten voor de communicatie tussen ODROID en server

In deze appendix worden de resultaten getoond waarvan de tests beschreven staan in sectie 4.1.

A.1 Verbinding tussen ODROID en server

A.1.1 Simpele test met server

```
$ ./client solarpoweredbikes.tudelft.nl 80
Please enter the message: client speaking
HTTP/1.1 400 Bad Request
Server: nginx
Date: Tue, 07 Jun 2016 12:24:42 GMT
Content-Type: text/html
Content-Length: 166
Connection: close

<html>
<head><title>400 Bad Request</title></head>
<body bgcolor="white">
<center><h1>400 Bad Request</h1><
```

A.1.2 Test met de correcte headers

```
$ ./client-php solarpoweredbikes.tudelft.nl 80
The message is:
POST /testInput.php HTTP/1.1
Host: solarpoweredbikes.tudelft.nl
Connection: Close
Content-Type: application/json
content-length: 19

{"fn":"A","sn":"B"}

HTTP/1.1 200 OK
Server: nginx
Date: Tue, 07 Jun 2016 12:40:46 GMT
Content-Type: text/html
Content-Length: 64
Connection: close
Set-Cookie: PHPSESSID=bd3sessr9jnaf4c0g3pdbv7ij7; path=/
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre
    -check=0
Pragma: no-cache
X-Powered-By: PleskLin

<html>
        <body>
                <!-- empty page! -->

```

```
27
28
29          </body>
30 </html>
```

A.2 JSON

A.2.1 Test waarbij JSON op de server gedecodeerd kan worden

```
1 $ ./client-php solarpoweredbikes.tudelft.nl 80
2 The message is:
3 POST /testInput.php HTTP/1.1
4 Host: solarpoweredbikes.tudelft.nl
5 Connection: Close
6 Content-Type: application/json
7 content-length: 19
8
9 {"fn":"A","sn":"B"}
10
11 HTTP/1.1 200 OK
12 Server: nginx
13 Date: Tue, 07 Jun 2016 14:10:47 GMT
14 Content-Type: text/html
15 Content-Length: 193
16 Connection: close
17 Set-Cookie: PHPSESSID=r4deocc79lfnminrip74hodgg6; path=/
18 Expires: Thu, 19 Nov 1981 08:52:00 GMT
19 Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre
     -check=0
20 Pragma: no-cache
21 X-Powered-By: PleskLin
22
23 <html>
24          <body>
25                  {"fn":"A","sn":"B"} is op de input verschenen<br>
26 decode: stdClass Object
27 (
28     [fn] => A
29     [sn] => B
30 )
31 <br>
32 Content-length:19<br>
33 Input size:19<br>
34
35
36
37          </body>
38 </html>
```

A.2.2 Test met encoderen/decoderen aan de C-zijde

```
1 $ ./test
2 Voer een paar getallen in, gescheiden door enters
3 Met q of EOF wordt het invoeren afgesloten
4 2
```

```
5 3
6 4
7 8
8 q
9
10 1
11 2
12 3
13 4
14 55
15 6
16 2
17 3
18 4
19 8
```

A.3 Valgrind

```
1 $ valgrind --tool=memcheck ./test
2 ==27453== Memcheck, a memory error detector
3 ==27453== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et
    al.
4 ==27453== Using Valgrind-3.10.1 and LibVEX; rerun with -h for
   copyright info
5 ==27453== Command: ./test
6 ==27453==
7 1
8 0
9 1
10 0
11 1
12 1
13
14 ==27453==
15 ==27453== HEAP SUMMARY:
16 ==27453==     in use at exit: 0 bytes in 0 blocks
17 ==27453==   total heap usage: 80 allocs, 80 frees, 9,201 bytes
   allocated
18 ==27453==
19 ==27453== All heap blocks were freed -- no leaks are possible
20 ==27453==
21 ==27453== For counts of detected and suppressed errors, rerun with: -
   v
22 ==27453== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from
    0)
```