Lectoraat Smart Energy

RAPPORT
IoT Infrastructuur Smart Energy Delivery Lab (SEnD Lab)

AE&I - Technische Informatica

datum : 21 juni 2019
versie : 0.1
status : Draft

auteur(s) : Maarten Nieuwenhuize, Lars Moesman, Rick Verstraten, Aaron Israëls, Diederich Kroeske en Maurice Snoeren

Management Samenvatting

// Diederich en Maurice

Inhoudsopgave

1 Inleiding

// Diederich en Maurice

1.1 Lectoraat Smart Energy

1.2 Achtergrond

1.3 Probleemstelling

1.4 Doelstelling

2 API

In dit hoofdstuk wordt de API van het SendLab besproken. In het SendLab bevinden zich meerdere sensoren waarvan de data moet worden opgeslagen in een database. Hiervoor is een koppeling nodig die de verschillende onderdelen verbindt. De API is het middelpunt van het SendLab en maakt het mogelijk dat alle andere onderdelen (sensoren, servers, database) met elkaar kunnen communiceren.

2.1 Team

Het API-Team bestond uit 4 personen: Maarten Nieuwenhuize, Aaron Israëls, Rick Verstraten en Lars Moesman. Samen hebben zij gewerkt aan de API van het SendLab.

Maarten: "Ik heb me voornamelijk gericht op het instellen van de mosquitto broker en de bijbehorende certificaten. Het opslaan van de inkomende data via de database-link applicatie. Ook heb ik gewerkt samen met andere groepsgenoten aan de event-generator-handler en event-logger applicatie. Ook heb ik gewerkt aan de zonneboot-websocket applicatie en de rest-full applicatie."

Aaron: "Ik heb me niet specifiek gericht op een taak maar heb me meer gericht op een aanvullende rol terwijl de rest zich vooral op één onderwerp had gestort. Heb o.a. meegewerkt aan het Swagger ontwerp, de event generator en bijgesprongen wanneer dit gewenst was. Richting het einde ben ik me ook vooral bezig gaan houden met documentatie en aanvullende documenten."

Rick: "Ik heb me voornamelijk gericht op de authenticatie van de API, zo heb ik de OAuth service ontworpen en geïmplementeerd. Hiernaast ben ik ook veel bezig geweest met het documenteren van de API in Swagger, zodat de API later gemakkelijk te gebruiken zou zijn door het visualisatie team. Tot slot heb ik geholpen met het uitwerken van de event logger applicatie."

Lars: "Ik ben voornamelijk bezig geweest met het opzetten van de Docker containers, zodat dat we lokaal makkelijk onze software konden testen en later naar de server te kunnen pushen. Verder heb ik ook een aantal scripts gemaakt om het uploaden naar de server makkelijker te maken voor het team. Daarnaast heb ik ook geholpen met een deel van de event handler en heb ik een bijdrage geleverd aan de event logger applicatie om uit te zoeken hoe ElectronJS werkt en om het te laten draaien op een Raspberry Pi."

2.2 Opdrachtbeschrijving

Binnen en buiten het SendLab staan verschillende IOT-apparaten die belangrijke data genereerd. Deze data zal volledig opgeslagen en verwerkt moeten worden. Hiervoor zal er een API ontwikkeld moeten worden die het mogelijk maakt de opslag aan de sensoren te linken.

Aan de opdrachtnemers is het de taak om een API op te zetten tussen de sensoren en de dataopslag. Deze API moet ook de mogelijkheid bieden om evenementen te genereren aan de hand van de inhoud van data die bij de API binnen komt.

2.3 Doelstellingen

Bij het ontwikkelen van het product zijn de volgende doelstellingen gesteld:

- Een API ontwikkelen die geschikt is voor meerdere dataformaten.
- Een goed gedocumenteerde API.
- Data wordt veilig getransporteerd.
- Event-based data.

2.4 Onderzoeksvraag

Hoofdvraag:

Hoe kan data van verschillende databronnen gecombineerd worden zodanig dat eindgebruikers deze data gemakkelijk kunnen verwerken waarbij events gegenereerd kunnen worden afhankelijk van de inhoud van de data?

Deelvragen:

1) Hoe kan het dataverkeer beveiligd worden?
2) Hoe wordt de data beschikbaar gesteld?
3) Welke standaarden zijn het best voor het versturen van data van IoT-apparaten?
4) Met welke huidige & toekomstige IoT-apparaten moet de APİ kunnen communiceren?
5) Waar gaat de APİ op draaien?
6) Hoe wordt de APİ gekoppeld aan de database(s)?
7) Met welke eisen van andere groepen moet er rekening gehouden worden?
8) Wanneer vinden events plaats?
9) Hoe kunnen events geïmplementeerd worden?

2.5 Aanpak

Voordat het project van start is gegaan heeft het team een plan van aanpak opgesteld, zie bijlage PVA API. In de bijlage is beschreven hoe dit project is aangepakt en de daarbij opgestelde planning.

De eerste 2 weken zijn ze van start gegaan met het opzetten van een MQTT-broker, waar alle apparaten zoals de Zonneboot en de Lotus hun data naar toe konden sturen. Omdat de servers een aantal weken later kwamen dan gedacht en het team geen dubbel werk wilde uitvoeren, is ervoor gekozen om te kijken naar de mogelijkheden van Docker.

Docker zorgt er namelijk voor dat er een omgeving wordt gecreëerd waarin alle benodigde software kon worden opgezet zoals de MQTT-broker. Deze omgeving kon later zonder problemen worden gedraaid op de servers.

Om de MQTT-broker te kunnen testen is er een applicatie gemaakt die een IoT-device simuleert. Deze is later ook nog van pas gekomen bij het testen van de Event Handler.

Na het opzetten van MQTT in Docker is er gewerkt aan de API voor het ophalen van de data. Voor het ontwikkelen van de API zijn eerst de benodigde endpoint opgezet, deze zijn later gekoppeld aan de database. Ook was het van belang om de data veilig over te sturen, daarom is er ook gebruik gemaakt van SSL-certificaten om de data de encrypten. Als laatst is de documentatie van de API in orde gemaakt zodat het visualisatie team met de API aan de slag kon.

Een van de eisen aan de API was dat deze real-time een bericht (event) zou kunnen genereren als er iets mis zou zijn met de binnenkomende data bijvoorbeeld. Hiervoor is de event handler applicatie ontwikkeld. Deze applicatie draai net als de REST-API in Docker en doorzoekt de binnenkomende data op eventuele fouten of als er geen data is de missende apparaten. Om deze berichten zichtbaar te maken is er een event logger applicatie in ElectronJS gemaakt. Dit was niet opgenomen in het Plan van Aanpak, maar het maakte het testen een stuk makkelijker, zo kon er namelijk met behulp van een Raspberry Pi en de event logger applicatie op een extern scherm de berichten worden getoond.

Om te voorkomen dat data publiek kon worden opgehaald is er een eigen OAuth service opgezet. Voor de OAuth service is gekozen een Node applicatie te schrijven. Ook deze applicatie is in een Docker container gestopt. Er is gekozen om de OAuth server op te zetten met drie endpoints, namelijk: "/login", "/refresh" en "/changepassword".

In de laatste weken is de koppeling met de database gemaakt met behulp van het database team, zodat de binnenkomende data eindelijk opgeslagen kon worden en natuurlijk later zou kunnen worden teruggevraagd via de API endpoints. De database is daarvoor ook in Docker komen te staan. Als laatst is er ook een Docker container gemaakt voor de Angular applicatie van het Zonneboot-Visualisatie team.

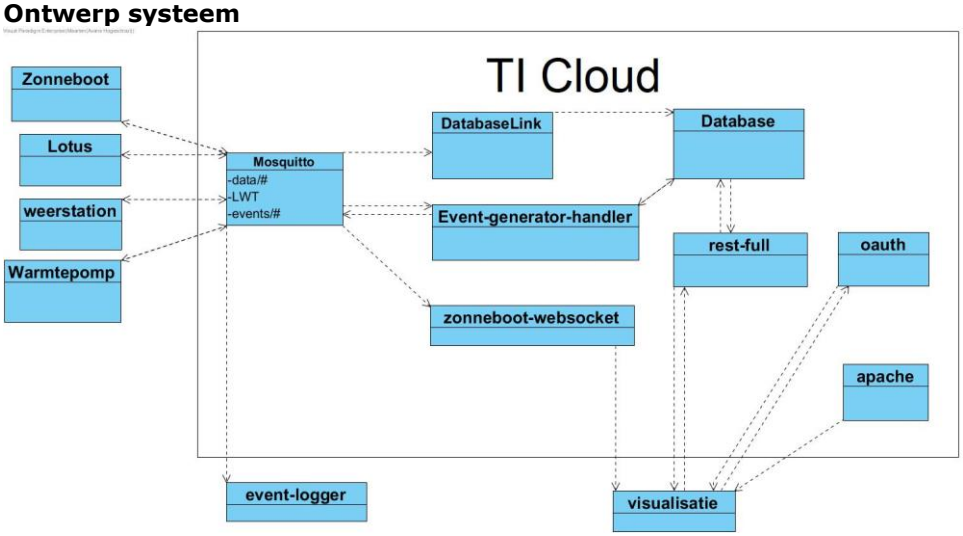
2.6 Onderzoeksresultaten

Een van de vele dingen die het team heeft gedaan is enorm nuttig gebleken, namelijk het gebruik van Docker. Docker is een applicatie die omgevingen creëert waarin je bepaalde services kan runnen. Docker is enorm handig omdat het, in tegenstelling tot een virtual machine, bijna niks aan computerkracht vereist. Docker zorgt er namelijk voor dat alleen het minimale van een operating system wordt gebruikt om services te kunnen runnen. Het makkelijke van Docker is daarnaast ook dat iedereen binnen (en buiten) het team kan testen op een systeem dat bestaat uit de juiste hulpmiddelen, hiermee voorkom je de situatie “het werkt wel op mijn pc”. Als alles eenmaal succesvol is getest is de Docker ook heel makkelijk op een server te deployen zodat de omgeving online komt en diensten kan aanbieden zoals bijvoorbeeld een webserver.

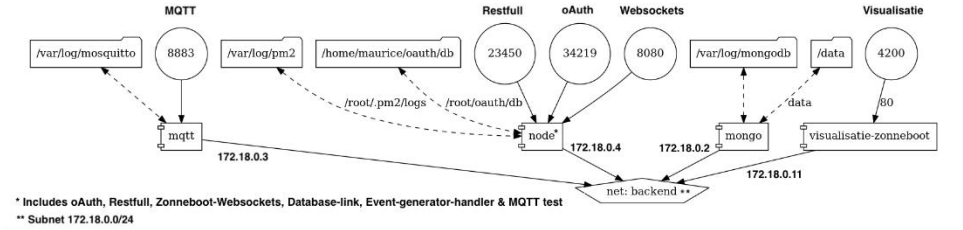
Ook het uitwerken van de OAuth service in een losse applicatie was zinvol. Door de authenticatie in een lost staan project te programmeren is het op een later moment namelijk gemakkelijk om bijkomende API’s te beveiligen. Hiernaast is OAuth de security van de API ten goede gekomen. Door gebruik te maken van OAuth hoeft de gebruiker minder vaak zijn logingegevens in te vullen. Hierdoor kunnen deze gegevens dan ook minder vaak onderschept worden. De OAuth service werkt als volgt. Nadat de gebruiker zichzelf heeft aangemeld ontvangt hij een access- en refreshtoken van de server. Het accesstoken dient bij API-calls meegegeven te worden in de Authorizationheader. De API zal vervolgens dit token valideren. Indien het token verlopen is wordt er een error teruggestuurd. Wanneer het accesstoken is verlopen dient een client met het refreshtoken een nieuw token op te vragen bij de OAuth service.

2.7 Ontwerp

Ontwerp systeem



Ontwerp Docker



Ontwerp API

Er is een ontwerp gemaakt voor het JSON object waarin alle sensordata wordt verpakt die verstuurt wordt over de API. Dit ontwerp is de bekijken in **Bijlage B**. Er is bewust gekozen een optioneel buffer veld toe te voegen omdat niet alle IOT-devices synchroon hun data uitzenden.

Er is ook een ontwerp gemaakt voor de event-generator met de bijbehorende events. Dit ontwerp is ook terug te zien in **Bijlage B**.

Alvorens het realiseren van de rest-full API is deze ontworpen. Hiervoor is gebruik gemaakt van Swagger. In dit ontwerp zijn alle endpoints opgenomen met hun bijbehorende parameters. In verband met security is gekozen om het verborgen "/changepassword" endpoint niet te documenteren.

Ontwerp van de API-structuur van het SendLab netwerk:
https://app.swaggerhub.com/apis/Rick-Farstreet/SendLab_rest/1.0.0#/default/in

2.8 Resultaten

Aan het einde van periode 3.4 is de API af. De structuur uit het ontwerp is volledig gerealiseerd. De volgende zaken zijn opgezet/ontwikkeld voor de API:

- MQTT: MQTT is opgezet om de data van de IoT devices naar de server te krijgen. De data wordt geëncrypt met behulp van SSL-certificaten,
- Event generator: De event generator genereerd een bericht naar aanleiding van de inhoud of het ontbreken van data die een IoT-device verstuurd.
- REST API: de REST API maakt het mogelijk om data op te vragen vanuit de frontend. Ook deze encrypt de data met behulp van SSL-certificaten.
- OAuth: OAuth is opgezet om de endpoint te beveiligen. Gebruikers van een endpoint moeten inloggen met een gebruikersnaam en wachtwoord, vervolgens krijgen ze tokens om de endpoints te kunnen gebruiken. Ook de data die naar de OAuth service wordt gestuurd is geëncrypt.
- MQTT-Test-App: Deze app is ontwikkeld om een IoT-device te kunnen simuleren. Zo kan de MQTT-broker getest worden en andere applicaties die IoT data gebruiken.
- Event Logger: De event logger is een app om de berichten die de event generator verstuurd te kunnen bekijken. Deze app is ook een mooie toevoeging geworden aan het project om te zien of alles goed gaat.

Alle backend applicaties worden gehost in een eigen Docker container. Op deze manier kunnen services en applicaties makkelijk getest en gedeployed worden. De Docker containers voorkomen daarnaast dat een fout grote impact heeft op het OS van de server. Als er iets fout gaat deploy je opnieuw de Docker container.

2.9 Conclusies

Al met al is het project dus goed verlopen. Alle applicaties en diensten die in het plan van aanpak waren opgenomen zijn uitgewerkt. Door Docker hebben we toch de eerste paar weken kunnen werken aan bijvoorbeeld MQTT ondanks dat er geen servers waren. Naderhand heeft dit ook geholpen bij het testen van de applicaties. Naast de applicaties en diensten die voorzien waren in het PVA is er ook tijd geweest om een extra applicatie te maken om het testen wat gemakkelijker te maken, de event logger. Het project heeft dus goed uitgepakt en het team is tevreden over het product.

2.10 Aanbevelingen

Nu het project voor ons tot een einde is gekomen hebben we een paar aanbevelingen:

- OAuth uitbreiden, heeft nu maar 1 gebruiker. Geen mogelijkheid om te registreren.
- Meer events voor de event handler.
- Meer endpoints voor de database.
- Meer schema’s toevoegen voor odd value event.
- Docker beter structureren. Makkelijker maken om losse applicaties te updaten ipv alle applicaties stil te moeten leggen.
- Documentatie bijhouden, de documentatie is nu helemaal compleet en het is ook handig om deze bij uitbreiding te blijven aanvullen.
- Versie nummers voor alle applicaties. Het is nu onduidelijk welke versie er is gedeployed op de server. Versienummer voorkomt verwarring.
- Unit tests maken voor de applicaties.

2.11 Relevante bronnen en documenten

Swagger documentatie van de API:
https://app.swaggerhub.com/apis/Rick-Farstreet/SendLab_rest/1.0.0#/default/in

Bijlage A: “PVA API”

PVA API

Aaron Israëls, Lars Moesman, Rick Verstraten en Maarten Nieuwenhuize | R&d | 04/04/2019 | V 1.0

Inleiding

1. Achtergronden

Project: Sendlab API
Projectnaam: APİ
Opdrachtgever: Avans AE&I
Opdrachtnemers: Aaron Israëls, Lars Moesman, Maarten Nieuwenhuize, Rick Verstraten

Het Sendlab is een project ter bevordering van de duurzaamheid van Avans Hogeschool Breda. In dit project worden meerdere sensoren en actuatoren aan elkaar gekoppeld. Alle data uit het Sendlab zal opgehaald en verstuurd moeten kunnen worden naar een centrale database. Om deze communicatie tussen de database en de end-user devices te realiseren is aan de opdrachtnemers de opdracht voorgelegd om een Application Programming Interface op te zetten waarmee deze apparaten eenvoudig en veilig kunnen communiceren.

2. Projectresultaat

Aanleiding

Binnen en buiten het SendLab staan verschillende IOT-apparaten waar belangrijke data uitkomt. Deze data zal volledig opgeslagen en verwerkt moeten worden. Hiervoor zal er een API ontwikkeld moeten worden.

Gewenst resultaat

Aan de opdrachtnemers is het de taak om een interface op te zetten tussen de clients en de dataopslag. Deze interface moet ook de mogelijkheid bieden om events te genereren aan de hand van de inhoud van data die bij de interface binnen komt.

Hoofdvraag

Hoe kan data van verschillende databronnen gecombineerd worden zodanig dat eindgebruikers deze data gemakkelijk kunnen verwerken waarbij events gegenereerd kunnen worden afhankelijk van de inhoud van de data?

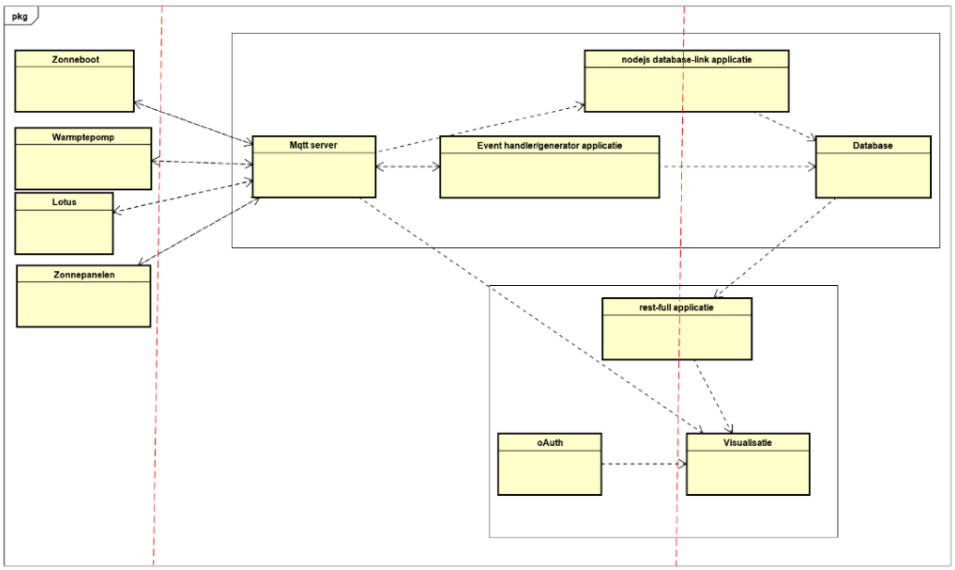
Deelvragen

1. Hoe kan het dataverkeer beveiligd worden?
2. Hoe wordt de data beschikbaar gesteld?
3. Welke standaarden zijn het best voor het versturen van data van IoT-apparaten?
4. Met welke huidige & toekomstige IoT-apparaten moet de APİ kunnen communiceren
5. Waar gaat de APİ op draaien?
6. Hoe wordt de APİ gekoppeld aan de database(s)?
7. Met welke eisen van andere groepen moet er rekening gehouden worden?
8. Wanneer vinden events plaats?
9. Hoe kunnen events geïmplementeerd worden?

Doelstellingen

- Een APİ ontwikkelen die geschikt is voor meerdere dataformaten.
- Een goed gedocumenteerde APİ.
- Data wordt veilig getransporteerd.
- Event-based data

Systeem



Voor de structuur van het systeem is er het bovenstaande design gemaakt. De IOT-devices draaiend op het LAN/wifinetwerk van het SendLab, verbinden met een server draaiend op het utiltiy-netwerk. Van buiten zal ook de zonneboot kunnen verbinden met deze server. De informatie die verstuurd wordt gaat over SMQTT (secure MQTT). Deze keuze is gemaakt omdat deze werkwijze al is behandeld in de lessen en de data uit de IOT-apparaten op meerdere plekken benodigd zal zijn. Er was een overweging gemaakt om wellicht een rest-full applicatie te gebruiken, maar deze werkwijze bleek een stuk ingewikkelder wegens de meervoudige implementaties van de data.

Op de server zal Ubuntu gedraaid worden. Deze keuze is gemaakt wegens eerdere ervaring met Ubuntu. Op deze server zal de MQTT-broker draaien. Daarbij is gekozen voor Mosquitto. Over Mosquitto is veel informatie beschikbaar en dit programma beschikt over alle functionaliteiten die nodig zijn voor dit project. MQTT is een protocol specifiek gemaakt voor low-power apparaten. Dit betekent dat MQTT minder energie verbruikt dan HTTPS posts. Ook heeft MQTT een last-will feature, deze feature zorgt ervoor dat zodra een client offline gaat er een bericht wordt gepost op een bepaalde topic. Hiermee kan er een event worden gegenereerd zodra hier een bericht van wordt gestuurd. Ook is er gekozen voor MQTT omdat de events die gegenereerd zullen worden op verschillende plaatsten nodig zijn.

Op de MQTT-broker zullen 3 topics gebruikt worden: data, events en LWT. Over het data topic gaat de data die van de IOT-apparaten komt (Zie API-document). De subscribers op deze topic zijn de database-link applicatie en de event handler/generator applicatie. Op de event topic worden de gegenereerde events verstuurd. De subscribers op deze topic zijn alle IOT-apparaten en de Visualisatie. Ook kunnen er in de toekomst applicaties op deze topic subscriben die bijvoorbeeld een sms stuurt naar een operator als een bepaald event wordt geroepen.

Op de server draaien ook 2 Node.js applicaties: een event-generator/handler applicatie en een database-connectie applicatie. De keuze voor Node.js is gemaakt omdat er binnen de

projectgroep veel ervaring hiermee heeft. Ook biedt Node.js de mogelijkheid om gemakkelijk MQTT-data binnen te halen en het te kunnen interfacen van verschillende databases.

De database-connectie applicatie wordt gedeeld en ontwikkeld, samen met het databaseteam. Binnen deze applicatie zijn de opdrachtnemers verantwoordelijk voor het binnenhalen van de IOT-data. Het API-team zorgt dat deze applicatie verbindt met de MQTT-broker en dat de binnengekomen data aan het databaseteam wordt gegeven. Het API-team geeft aan het database team een record met altijd: identiefier (waar de data vandaan komt), timestamp en de data zelf. Het database team is daarna verantwoordelijk voor het opslaan van de IOT-data in de database.

De event-generator/handler applicatie is verantwoordelijk voor het detecteren van events. Deze events zijn gespecificeerd in het events-API-document. Daarna wordt het event verstuurd over de event topic op de MQTT-broker.

Om de opgeslagen data beschikbaar te stellen voor het visualisatie team, zal er een rest-full applicatie opgezet worden. REST maakt het mogelijk om een API op te zetten. De ondersteuning voor meerdere datastructuren, waaronder JSON, maakt REST geschikt voor de API. Doordat REST JSON ondersteund is het makkelijk om de API te gebruiken op meerdere platformen, omdat (bijna) elk platform om kan gaan met JSON-objecten. Naast de ondersteuning van JSON, is een rest-full API een efficiënte manier om data van de backend naar de front-end te krijgen.

Binnen het DMZ-netwerk bevindt zich ook een draaiende Ubuntu server. Op deze server komt de rest-full applicatie te draaien die de informatie uit de database online zet. Deze applicatie zal ook gedeeld worden met het database-team. Het database-team is verantwoordelijk voor het ophalen van de data uit de database(query's). Het API-team is verantwoordelijk voor het maken van end-points naar deze data.

Voor het beveiligen van de rest-full applicatie wordt gebruik gemaakt van OAuth. Op de server zal een OAuth authenticatie server draaien (Node.js). Op deze server kan worden ingelogd waarna de gebruiker een refresh- en accestoken krijgt. Bij iedere request zal de accesstoken in de Authorization header meegestuurd worden naar de rest-full server. De rest-full server kan daarna verifiëren of de accestoken gegeven is door de OAuth-server. Na één uur zal de accestoken vervallen en zal de eindgebruiker de refreshtoken naar de OAuth-server moeten sturen. De OAuth-server stuurt dan een nieuwe refresh- en accestoken terug. De rest-full server en de OAuth-server zullen tijdens dit proces niet met elkaar communiceren.

3. Projectactiviteiten

Onderzoek beveiliging dataverkeer

Overleg met andere groepen

- Overleg datastructuur
- Overleg koppeling database
- Overleg verschillende IoT apparaten die gekoppeld moeten worden
- Overleg events

Ontwerpen structuur APİ

- Ontwerpen event systeem
- Ontwerpen REST APİ

Realiseren APİ

- Maken document datastructuur
- MQTT Server opzetten met TLS
- Virtual IoT device
- Node.JS database-link opzetten
- REST API ontwikkelen
- Event systeem ontwikkelen
- OAuth implementeren
- Swagger
- REST-full generated test data
- Beveiliging implementeren
- Unit tests opzetten

APİ Testen

APİ Koppelen aan database

APİ Documenteren

4. Projectgrenzen & Randvoorwaarden

De projectgroep is verantwoordelijk voor:

- Het maken van een JSON-standaard die verstuurd moet worden over SMQTT.
- Het maken van een SMQTT server met werkende certificaten.
- Het verbinden van de data van sensoren met de database.
- Genereren van events.
- Data beschikbaar stellen aan eindgebruikers.
- Het ophalen van de IOT-data en deze data doorgeven aan het database-team
- Het maken van endpoints voor de rest-full applicatie aan de hand van de data van het databaseteam

De projectgroep is *niet* verantwoordelijk voor:

- Opslag van gegenereerde data.
- Uitlezen van sensoren.
- Weergave van data.
- Netwerkverbindingen van IOT-apparaten.

5. Tussenresultaten

Analyse & ontwerp

Aan het einde van de eerste periode zal door de projectgroep worden opgeleverd:

- Plan van Aanpak
- Github ingericht
- Trello pagina

Realisatie & Testen

Aan het einde van de tweede periode zal door de projectgroep worden opgeleverd:

- SMQTT broker
- API-documentatie
- Virtual IoT device
- IOT-data kan worden ophaalt in de databaselink-applicatie.
- Events kunnen worden gemaakt aan de hand van de IOT-applicatie.
- De data die database wordt gehaald door het database-team is gekoppeld aan endpoints
- De rest-full applicatie is beveiligd d.m.v. OAuth
- Swagger
- REST-full generated test data
- Unit tests.

6. Kwaliteit

Het project zal ontwikkeld worden met behulp van de SCRUM-methode. Aan het begin van elke sprint zal er besproken worden welke user story's er in de desbetreffende sprint uitgewerkt zullen worden. De user story wordt dan vertaald naar taken die op het SCRUM-bord (Trello) komen te staan.

Elke ontwikkelaar zal bij het ontwikkelen van een desbetreffende taak er zelf voor moeten zorgen dat de functionaliteit goed getest wordt voordat een taak als voltooid beschouwd mag worden. De laatste dag van de sprint zal er dan voor gezorgd worden dat alle functionaliteiten worden samengevoegd en het geheel een keer getest wordt.

Elke ochtend zal er een SCRUM-meeting plaatsvinden waarin de ontwikkelaars elkaar op de hoogte zullen brengen over het project. Hier zal besproken worden wat iedereen die dag gaat doen en of er eventueel nog problemen zijn waar rekening mee gehouden moet worden.

Daarnaast zullen er duidelijke afspraken gemaakt moeten worden met andere groepen over hoe de APİ eruit gaat komen te zien. Zodat het implementeren van de APİ soepel zal verlopen.

Op deze 2 manieren zullen de opdrachtnemers proberen de kwaliteit van het product te waarborgen.

7. Projectorganisatie

Contactinformatie

TEAM API

Lars Moesman	
Functie/rol binnen project	**Opdrachtnemer**
Mail	**l.moesman@student.avans.nl**
Telefoon	**06 12 41 77 81**

Rick Verstraten	
Functie/rol binnen project	**Opdrachtnemer**
Mail	**rcj.verstraten@student.avans.nl**
Telefoon	**06 34 60 18 28**

Maarten Nieuwenhuize	
Functie/rol binnen project	**Opdrachtnemer**
Mail	**mw.nieuwenhuize@student.avans.nl**
Telefoon	**06 21 61 15 66**

Aaron Israels	
Functie/rol binnen project	**Opdrachtnemer**
Mail	**ajk.israels@student.avans.nl**
Telefoon	**06 37 43 69 59**

TEAM Database

Kevin Quist	
Functie/rol binnen project	**Software Engineer Database**
Mail	**cm.quist@student.avans.nl**

Ricky Hoogerdijk	
Functie/rol binnen project	**Software Engineer Database**
Mail	**rla.hoogerdijk@student.avans.nl**

TEAM Solarboat

Lois Gussenhoven	

Functie/rol binnen project	**Software Engineer Solarboat**
Mail	**lly.gussenhoven@student.avans.nl**

Nick van Endhoven	
Functie/rol binnen project	**Software Engineer Solarboat**
Mail	**n.vanendhoven@student.avans.nl**

Gijs van Fessem	
Functie/rol binnen project	**Software Engineer Solarboat**
Mail	**gcm.vanfessem@student.avans.nl**

Cas Koopmans	
Functie/rol binnen project	**Software Engineer Solarboat**
Mail	**pmans@student.avans.nl cjw.koo**

TEAM Lotus

Sam Nicknam	
Functie/rol binnen project	**Software Engineer LOTUS**
Mail	**s.nicknam@student.avans.nl**

Thijs Wijnen	
Functie/rol binnen project	**Software Engineer LOTUS**
Mail	**ma.wijnen2@student.avans.nl**

Thomas Brouwers	
Functie/rol binnen project	**Software Engineer LOTUS**
Mail	**tm.brouwers@student.avans.nl**

TEAM Warmtepomp

Dion Bartelen	
Functie/rol binnen project	**Software Engineer Warmtepomp**
Mail	**agm.bartelen@student.avans.nl**

Jeffrey Lantinga	
Functie/rol binnen project	**Software Engineer Warmtepomp**
Mail	**j.lantinga@student.avans.nl**

8. Planning

Week	Activiteit
1	• Definiëren acceptatietests • MQTT Broker opzetten • TLS voor MQTT-server opzetten • Verder overleg andere groepen
2	• Opzetten virtual IoT-device • Data-link applicatie implementeren MQTTS • Link data naar databaseteam
3	• REST API-interface opzetten • Router links opzetten • Opstarten OAuth
4	• Verder OAuth • REST test opzetten • Swagger implementeren
5	• Verder overleg andere groepen over events • Event systeem ontwikkelen • Verder uitwerken REST test voor events
6	• Acceptatie test uitvoeren • Implementatie starten
7	• Overleg andere groepen • Verder implementeren
8	• Overleg andere groepen • Verder implementeren
9	• Opleveren

9. Kosten & Baten

Kosten

- Aanschaf Servers
- Stroomkosten Servers
- Domein/ Certificaten

Baten

- Kennis over het opzetten van een API
- Energiebesparing binnen Avans Hogeschool Breda

10. Risico's

Risico	Risicokans	Gevolg	Oplossing
Langdurige ziekte	Niet voorspelbaar	Mogelijk wordt de planning niet behaald.	Op tijd melden bij groepsleden
Tijdsnood	Klein	Het eindproduct bevat niet alle gewenste functionaliteiten die opgesteld zijn	Een haalbare planning opstellen, zal ervoor zorgen dat dit niet of niet zo snel voorkomt.
Defecte laptop	Klein	Er kan niet gewerkt worden aan de opdracht en/of verslagen	Zo spoedig mogelijk een vervangende laptop regelen.
OV rijdt niet	Klein	Het is niet mogelijk om op de stageplek te komen.	Alternatief vervoer regelen, indien dit niet lukt de andere groepsleden inlichten dat je thuis aan de slag gaat om zo de planning te kunnen volgen.

BIJLAGE B: ONTWERPEN

```
"verion-api" : "1.0"                        [float]     //version number
"timestamp" : "[YYYY-MM-DD HH:MM:SS UTC]"   [string]    //timestamp
"identifier" : "warmtepomp-001"             [string]    //unique identifier of device
"buffer" : [        !optional               [array]     //array of the members of the data. Each member should be an array with records with timestamp and value's.
    "gps","motor"                           [string]    //This value should be filled if using the buffer construction
    ]
"data" : {                                              //De data is stored here. See below for more information!

}
sensors:[                                               //This is an list consisting of all the sensor the IOT-device is connected to.
    {
        "sensor-name": "true/false"         [boolean]   //When the IOT-device loses connection to a sensor it should set the value to false. If it reconnects is should be set to true.
    },
    {
        "sensor-name2": "true/false"
    }
]
```

Figuur 1: Ontwerp API-object.

```
{
    "version-events" : "1.0"                    [float]     //verion number of the events api
    "timestamp" : "[YYYY-MM-DD HH:MM:SS UTC]"   [string]    //timestamp
    "identifier" : "warmtepomp-001"             [string]    //unique identifier of IOT device
    events [                                                //list of all events that have been created
        type: ""                                [string]    //type of event see list below for extra info
        message: ""                             [string]    //message that is send with the type
    ]
}

{ //sensor-off event
    version-events: "1.0"                                   //verion number of the events api
    timestamp: "[YYYY-MM-DD HH:MM:SS UTC]"                  //timestamp
    identifier: "warmtepomp-001"                            //unique identifier of IOT device
    events [
        {
            type: "sensor-off"                              //this event gets called when a sensor has been turned off.
            message: "/sensor-name/ has turned off"
        }
    ]
}
{ //sensor-on event
    version-events: "1.0"                                   //verion number of the events api
    timestamp: "[YYYY-MM-DD HH:MM:SS UTC]"                  //timestamp
    identifier: "warmtepomp-001"                            //unique identifier of IOT device
    events [
        {
            type: "sensor-on"                               //this event gets called when a sensor has been turned off.
            message: "/sensor-name/ has turned on"
        }
    ]
}
{ //connection-lost event
    version-events: "1.0"                                   //verion number of the events api
    timestamp: "[YYYY-MM-DD HH:MM:SS UTC]"                  //timestamp
    identifier: "warmtepomp-001"                            //unique identifier of IOT device
    events [
        {
            type: "connection-lost"                         //this event gets called when the API hasn't received an message from an IOT-device in x seconds
            message: "/identifier/ has lost connection last call was at /last timestamp/"
        }
    ]
}
```

```
{ //onnection-reastablised event
    version-events: "1.0"                               //verion number of the events api
    timestamp: "[YYYY-MM-DD HH:MM:SS UTC]"              //timestamp
    identifier: "warmtepomp-001"                        //unique identifier of IOT device
    events [
        {
            type: "connection-reastablised"             //this event gets called when the API receives a message from an IOT-device after the connection has been lost
            message: "/identifier/ has reastablised connection, /identifier/'s previous call was at /last timestamp/"
        }
    ]
}
{ //value-odd event
    version-events: "1.0"                               //verion number of the events api
    timestamp: "[YYYY-MM-DD HH:MM:SS UTC]"              //timestamp
    identifier: "warmtepomp-001"                        //unique identifier of IOT device
    events [
        {
            type: "value-odd"                           //this event gets called when an value is odd
            message: "/identifier/ has weird value gps:{latitute:-888}[range=1,20]"
        }
    ]
}
{ //wrong-metadata event
    version-events: "1.0"                               //verion number of the events api
    timestamp: "[YYYY-MM-DD HH:MM:SS UTC]"              //timestamp
    identifier: "warmtepomp-001"                        //unique identifier of IOT device
    events [
        {
            type: "wrong-metadata"                      //this event gets called when data is received with wrong metadata
            message: "identifier /name of identifier/ tried to push wrong metadata: wrong version should be 1.0"
        }
    ]
}
{ //new-device event
    version-events: "1.0"                               //verion number of the events api
    timestamp: "[YYYY-MM-DD HH:MM:SS UTC]"              //timestamp
    identifier: "warmtepomp-001"                        //unique identifier of IOT device
    events [
        {
            type: "new-device"                          //this event gets called when an IOT-device pushes data for the first time
            message: "a new device has been added with identifier /identifier/"
        }
    ]
}
```

Figuur 2: Ontwerp Event-generator