2019-2020

Software architecture

Project

In vrijwel elke grote stad zijn deelsystemen beschikbaar om je flexibel te verplaatsen op een ecologisch verantwoorde manier. Je bouwt in dit vak software voor een dergelijk systeem.

Probleembeschrijving

Op verschillende plaatsen verspreid over de stad bevinden zich *station*s van verschillende grootte waarin '*station vehicles'* zijn verankerd in een elektronische *lock.* Een gebruiker kan bij een station een vehicle unlocken[1]. Het systeem bepaalt welke lock zich opent. Later kan hij dit vehicle weer inchecken bij een station door het in een door het systeem toegewezen lock te plaatsen.
Naast de vehicles waarmee je van het ene naar het andere station kan rijden zijn er ook 'free vehicles' zoals steps, scooters,... beschikbaar die je gewoon ergens kan oppikken en achterlaten. Zij hebben een elektronisch slot dat zichzelf los of vastzet wanneer een gebruiker via een app[2] aangeeft dat hij zijn rit wil starten/beëindigen.

Om van het systeem gebruikt te maken dient men een subscription te nemen. Afhankelijk van het subscription type (dag, week of jaar) en het vehicle type (station, step, scooter) wordt een gedeelte van de rit per minuut aangerekend[3]. De te gebruiken minuten/tarieven voor deze berekening worden door een extern systeem aangeleverd. De prijs wordt telkens bij het beëindigen van een rit berekend en aan een extern facturatiesysteem afgeleverd.

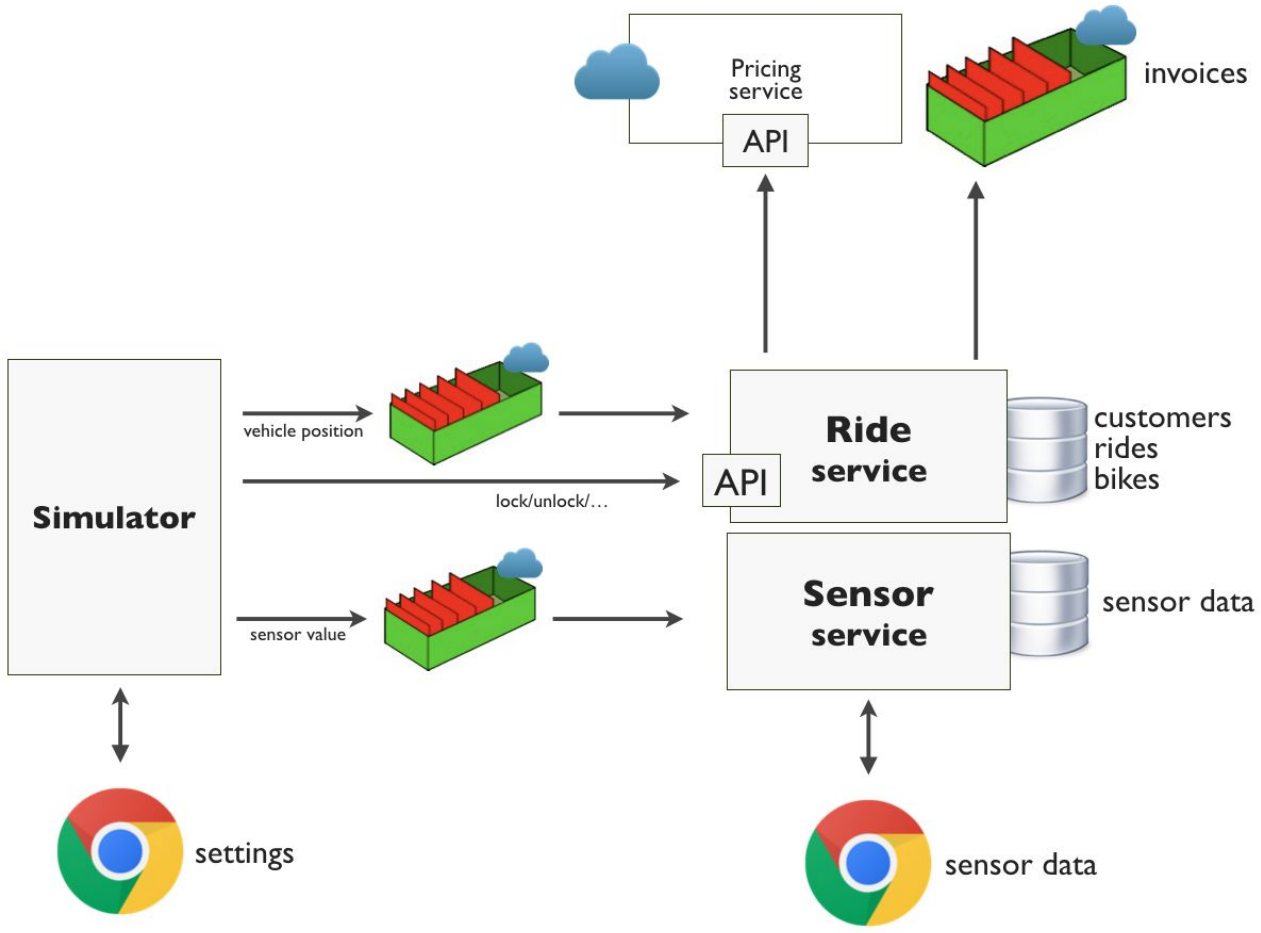
[1] Dit gebeurt door een pas voor een magneetlezer bij het station te houden
[2] Het schrijven van deze app zelf behoort niet tot de opdracht
[3] Bijvoorbeeld: met een jaarpas mag je met een station vehicle 20 minuten gratis rijden alvorens je het terug moet locken en betaal je 2 euro per bijkomende minuut

Free vehicles hebben een GPS die op geregelde tijdstippen de vehicle locatie doorstuurt naar het systeem. Dit laat toe om vehicles op locatie te zoeken, maar wordt ook gebruikt voor 'open ride' detectie. Een open ride komt voor wanneer een gebruiker een vehicle vergeet te locken na gebruik

Omdat ze zich continu doorheen de stad begeven maakt men gebruik van de vehicles om real-time luchtkwaliteitsmetingen (CO2, NO2, fijn stof,...) te vergaren. Verschillende types sensoren kunnen aan station of free vehicles worden bevestigd. Ze voeren metingen uit en sturen ze samen met de locatie door naar het systeem. Via een web interface kunnen deze metingen bekeken worden in grafieken.

Analisten en operationele medewerkers van de stad moeten het systeem kunnen testen en finetunen. Het ontwikkelen van een aparte toepassing die gebruikers en sensoren kan simuleren is daarom een essentieel onderdeel van het project. Deze simulator heeft een web interface voor het beheren van simulatie instellingen.

Volgende figuur geeft een overzicht van de globale architectuur van het systeem.



Functionaliteit

Simulator

In onderstaande tabel zijn de 'events' weergegeven die de simulator moet kunnen produceren. Sommige van deze events resulteren in een **call** naar de (REST) API van de RideService (zie verder), andere in het versturen van een **message** via een queue.

Event	Type	Params/Data	Return
unlock station vehicle	call	userID stationID	lockID
unlock free vehicle	call	userID vehicleID	true/false
location of free vehicle	message	timeStamp vehicleID xCoord yCoord	-
get free locks	call	stationID	list of lockID
lock station vehicle	call	userID lockID	-
lock free vehicle	call	vehicleID userID	true/false
find nearest free vehicle	call	xCoord yCoord vehicleType	vehicleID xCoord yCoord
sensor value	message	timeStamp xCoord yCoord type value	-

Het data formaat voor zowel de API als de queues is JSON. De precieze details van het formaat zijn vrij te kiezen. **USID20**

De simulator kan 2 types van simulaties draaien: ride en sensor. In een **ride simulatie** **USID15** worden ride events (dit zijn alle events in bovenstaande tabel behalve de laatste rij) uitgelezen uit een CSV file. Hierin staat per event een door komma's gescheiden lijn met de event data en een delay. De delay (in millis) bepaalt hoe lang er na het vorige event moet gewacht worden. **USID16** De simulator leest deze file en voert de nodige acties uit (API calls of location messages op de queue zetten). Er mag vanuit gegaan worden (indien dit niet zo is wordt dit als error gelogd) dat de files zodanig zijn opgebouwd dat ze 1 rit van 1 gebruiker bevatten (met een station -of een free vehicle).

Een **sensor simulatie** is bedoeld voor de generatie van sensor berichten (bv. voor performance testen) USID18. Deze worden niet uit een file gelezen maar random gegenereerd en op de sensor queue geplaatst volgens default instellingen die uit een configuratiebestand worden gelezen:

- tijdsperiode van generatie (bv. 60min)
- gemiddeld delay tussen de berichten
- grootte van random variatie op dit delay (bv. delay van 2000ms met variantie van 200ms geeft 1821, 2134,...)
- xCoord range (random generatie binnen deze grenzen)
- yCoord range (random generatie binnen deze grenzen)
- lijst van sensor types (er wordt random gevarieerd over de types)
    - type (bv. 'CO2')
    - value range voor dit type sensor (random generatie binnen deze grenzen)

De simulator heeft een web interface om deze te bedienen:

- USID17 Simuleren van een rit via upload van een CSV file USID47
- USID19 Starten van een sensor simulatie op basis van instellingen. De default instellingen worden getoond in form fields en kunnen worden aangepast door de gebruiker (ze hoeven niet opgeslagen te worden) USID48

Simulaties worden asynchroon gestart zodat de webpagina niet 'hangt' terwijl de simulatie bezig is.

Services

SensorService

Dit is een relatief eenvoudige toepassing die de inkomende sensor waarden uit de queue leest en opslaat in een databank USID7. Hiervoor mag een in memory H2 database gebruikt worden.

De gebruiker kan via een web interface de sensor data bekijken met volgende visualisaties:

- lijst van waarden USID6
- lijn-grafiek USID8
- heatmap (weergave op een kaart, de kleur wordt bepaald door de waarde op die locatie) USID9

Er kan gefiltered worden op type meting, tijd en locatie USID22

RideService

Deze service verwerkt de eigenlijke ritten en heeft volgende verantwoordelijkheden:

- **Aanbieden van een API** voor
    - het locken en unlocken van vehicles (zie hoger) USID1 USID2 USID3 USID4
    - het zoeken van free vehicles USID10
    - Het opvragen van free locks USID56
    - CRUD beheer van bikes USID23

- **Ontvangen van locations** van free vehicles via een queue en deze verwerken USID24

- **Facturatie**. Bij het afsluiten van een rit wordt berekend wat de gebruiker hiervoor dient te betalen USID12. Hiervoor wordt een *PricingService* gecontacteerd USID26 met het

subscription type en vehicle type als inputs. De service geeft op basis hiervan het aantal gratis minuten en de toeslag per minuut terug. Van elke ritprijs wordt een XML bericht (userID, prijs, ride details) gemaakt dat op een uitgaande queue wordt geplaatst. USID11

Een locale proxy (jar file) om de PricingService aan te roepen wordt ter beschikking gesteld. Tarieven veranderen echter slechts sporadisch. Het is dus zinvol om resultaten van aanroepen naar deze service te **cachen** USID27. Dit betekent dat het resultaat (tijdelijk) wordt bijgehouden en bij een volgende call (met dezelfde input parameters) de service niet aangeroepen wordt. Het vorige resultaat uit de cache wordt teruggegeven.

- **Open ride detectie**. Het kan gebeuren dat een gebruiker een *ride* niet goed afsluit aan het einde van een rit. Om oplopende kosten te vermijden is het belangrijk dat dit zo snel mogelijk wordt gedetecteerd zodat er een notificatie (je plaatst deze gewoon op de log) kan verstuurd worden. Er zijn 2 mechanismen:
  - **Tijd-gebaseerd** USID28
    Een rit die langer duurt dan x minuten (setting) wordt als open-ended beschouwd.
    Tijd-gebaseerde detectie wordt zowel voor free als station vehicles toegepast.
  - **Locatie-gebaseerd** USID29
    Voor free vehicles zal het achtergelaten vehicle positie informatie blijven uitzenden vanop dezelfde locatie. Door de beperkte nauwkeurigheid van GPS signalen zullen hier evenwel kleine fluctuaties op zitten. Als de verplaatsing gedurende y minuten (setting) onder een bepaalde fluctuatie z blijft (setting) wordt de rit als open-ended beschouwd. Locatie-gebaseerde detectie is enkel relevant voor free vehicles.

Voorzien zijn op mogelijke wijziging en uitbreiding

De code moet (met behulp van interfaces, patterns,...) open zijn voor toekomstige uitbreidingen

- USID52 andere methoden van prijsberekening voor een rit
- USID30 andere methoden om open rides te detecteren
- USID20 andere data formaten en protocollen

Change requests in de loop van het project zijn mogelijk.

Testen

Volgende geautomatiseerde testen worden voorzien:

- USID49 Unit test van de prijsberekening in de *RideService* met mocking van de *PricingService*
- USID50 Unit testen voor open ride detectie
- USID51 RideService API testen

Configuratie en Logging

USID32, USID4, USID46

De code wordt voorzien van zinvolle logging statements zowel op vlak van diagnose (debug, info) als foutafhandeling (warn, error...). Zaken die kunnen mislopen en dus gelogd moeten worden zijn onder meer:

- *De API wordt aangeroepen met foutieve input data* (bv. niet gekend vehicleID). Een error code (volgens REST conventies) wordt teruggestuurd en een warning gelogd.

- *De PricingService kan niet bereikt worden of geeft een fout terug*. De rit facturatie wordt op een later tijdstip opnieuw verwerkt. Een warning wordt gelogd.
- *Databases kunnen niet bereikt worden of geven onverwachte fouten terug*. Een error wordt gelogd en een error code teruggestuurd (indien de database toegang het gevolg was van een API call).
- *Queues kunnen niet bereikt worden of geven fouten terug.* Een error wordt gelogd. Indien het de uitgaande facturatie queue betreft wordt de rit op een later tijdstip opnieuw verwerkt.
- *Location of sensor berichten kunnen niet geconverteerd worden van stream formaat naar memory*. Het bericht wordt niet verwerkt. Een error wordt gelogd
- *CSV files kunnen niet correct ingelezen of verwerkt worden*. De file wordt niet verwerkt. Een error wordt gelogd

USID31, USID41, USID42

Queue en database settings (host,...) worden uit een configuratie bestand gelezen[4]

Security

Security valt buiten de scope van de applicatie. De API calls gebeuren dus zonder tokens en je hoeft niet aan te loggen op de web interfaces van Simulator en SensorService.
Application security komt ruim aan bod in Integratieproject 2.

Extra features

Volgende features kunnen in samenspraak met je coach worden opgenomen indien de basisopgave is afgewerkt:

- Verschillende ride CSV's in parallel verwerken in de Simulator
- Voor het opslaan van de sensor data gebruik maken van een databank die specifiek bedoeld is voor het opslaan van grote reeksen tijds/locatie data.
- Simulator geeft feedback over het verloop van gestarte simulaties
- Form based security op de Simulator web interface
- Token based security op de RideService API
- … (suggesties zijn welkom)

Technische specificaties

Je gebruikt **Spring Boot** (Java of Kotlin) en gradle voor de ontwikkeling. **USID52/53/54**

USID55

Je gebruikt een SQL Server 2017 database voor de rides/bikes/customers en een database naar keuze (bv. in-memory H2 database) voor de sensor values. De SQL Server database wordt aangeleverd als een drop/create/load script. Dit is ook de database die gebruikt wordt voor het project in het vak Databanken 3. Het is toegelaten (binnen het vak Software architecture) om deze database in een Docker container te draaien.

[4] application.properties in Spring terminologie

Voor de queue gebruik je CloudAMQP messaging **USID38** .

USID33/43/44
Het versiebeheer gebeurt in **Gitlab.** Ook de project planning en opvolging gebeurt aan de hand van **Gitlab**.

Zie de slides *Inleiding en Afspraken* en de infosessies voor verdere details over de opvolging, technologieën, architectuur etc..