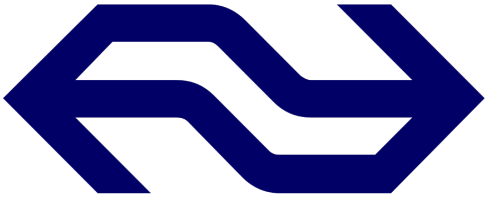
Technisch Ontwerp

Opdracht	Ritprofiel-analyse
Organisatie	Info Support
Opdrachtgever	Tim van den Hof, namens Nationale Spoorwegen
Schoolbegeleider	Paul Veldhuijzen van Zanten (VNP02)



Auteurs:

Mark den Boer	s1103928
Maikel Haarmans	s1102995
Willmar Knikker	s1102486
Nicander Mohrmann	s1103064
Silvan Otten	s1043664

Hogeschool Windesheim
ICT Software Engineering

Versie: 1.0
Datum: 17-01-2019

Samenvatting

Vanuit Info Support is er een vraag om de efficiëntie van machinisten van NS te verbeteren middels een app. Deze app wordt gebouwd met Angular, Ionic en Cordova als frontend in samenwerking met een dotnet core backend. Hierdoor draait de app dus cross-platform op iOS en Android.

Er is een bestaande API van Info Support waar logging data vandaan zal komen, maar hier maken wij in de scope van ons project geen gebruik van omdat wij gewoon logging data in CSV-formaat zullen laden.

De opdracht bestaat voor het grootste gedeelte uit het inzien van je rijstijl en deze kunnen analyseren, door deze te vergelijken met anderen en de ideale rit. Hierbij zit ook een onderdeel gamification om de gebruikers interactie te verhogen.

Dit document gaat verder in op de technische aspecten van de applicatie. Het geeft een overzicht hoe de app ingericht hoort te zijn en wat er nodig is dit te realiseren. Hierbij wordt de omgeving waarin het geplaatst is duidelijk en worden karakteristieken die de app heeft beschreven. Daarbij worden de relevante architectuur en infrastructuur toegelicht.

Vervolgens geven we de aan te houden technische keuzes om het product te ontwerpen en realiseren. Hierbij zijn de openstaande issues die nog opgepakt moeten worden bijgevoegd. Mogelijke verbeteringen voor in de toekomst volgen hierna.

Tot slot wordt er op de gebruikte componenten voor het gerealiseerde product ingegaan, ondersteund met modellen.

Inhoudsopgave

1 Inleiding

1.1 Doel van dit document

Dit document vloeit voort uit het bestaande Functioneel Ontwerp voor deze applicatie. Het is aan te raden voor de lezer van dit document om bekend te zijn met ons Functioneel Ontwerp aangezien deze twee documenten op elkaar aansluiten.
Dit document is opgesteld om te beschrijven hoe de app ontwikkeld zal worden.
Dit omvat onderdelen zoals: welke technieken er worden gebruikt, welke API calls er worden gebruikt, etc.
In dit document onderbouwen we onze gemaakte keuzes en ondersteunen wij eventuele developers die later aan dit project gaan werken. Ook wordt dit document gebruikt door de huidige devs om af te stemmen met elkaar over technische designkeuzes.

1.2 Scope

Aan het einde van dit project is het resultaat een proof of concept waarmee Info Support het project bij Nederlandse Spoorwegen zou kunnen voorleggen. Dit houdt in dat ons project een compleet plaatje moet zijn waar ook meerwaarde in te zien is.
Ons project houdt in dat een machinist verschillende gegevens kan inzien aan het einde van zijn gereden rit. Dit gebeurd door data te tonen over: De gereden rit (een grafiek die snelheid toont op de afstand die gereden is), score van de rit, statistieken van de rit en een advies over de rit.

1.3 Afhankelijkheden

Info Support heeft ons de keuze gegeven om een mobiele applicatie te maken in Cordova of Xamarin, met een backend waarbij dotnet core wordt gebruikt om alle berekeningen uit te voeren en deze met een API call beschikbaar te stellen voor de mobiele applicatie. Wij hebben gekozen om met Cordova onze mobiele applicatie te realiseren, omdat je met Cordova gemakkelijk cross-platform kan programmeren. Voor het ontwikkelen wordt in combinatie met Cordova ook Ionic gebruikt als framework om een goede gebruiksvriendelijke applicatie te realiseren.

1.4 Documentstructuur

1 Inleiding

Introductie die het project en dit document beschrijft.

2 Systeemoverzicht

Geeft een overzicht van het systeem in grote lijnen. Daarbij wordt de architectuur omschreven. Aangevuld met hoe er getest wordt.

3 Systeemontwerp

Geeft een overzicht van onze standaarden over documentatie en code, ook worden de gebruikte tools en openstaande issues hier behandeld.

4 Frontend

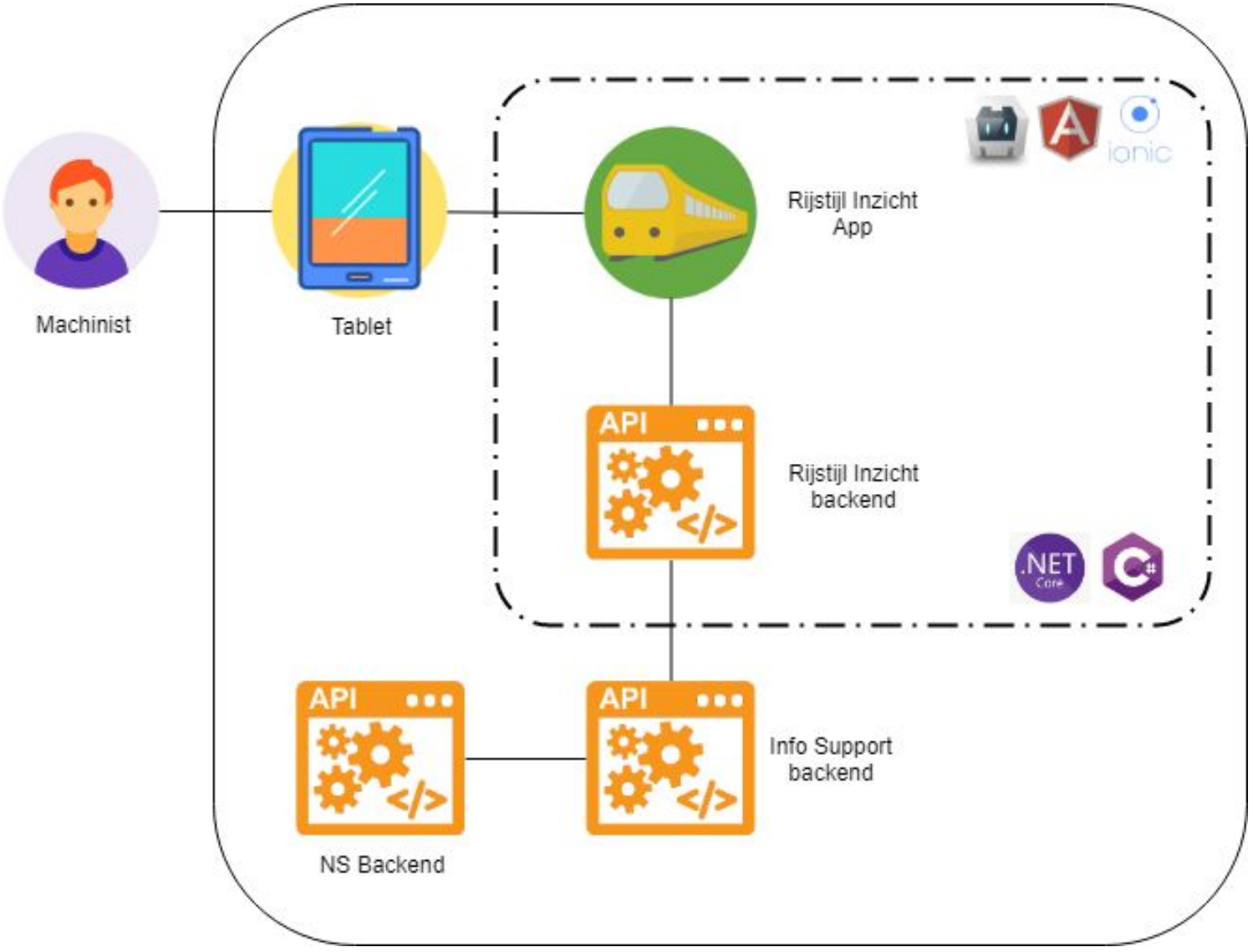
Toont een overzicht van de frontend en gaat in op individuele componenten.

5 Backend

Toont een overzicht van de backend en gaat in op individuele componenten. Aangevuld met de API-endpoints en Database informatie.

2 Systeemoverzicht

2.1 Context



De applicatie die wij gaan realiseren bestaat uit meerdere onderdelen, een Android/iOS applicatie om machinisten te stimuleren en een API gemaakt in C# om de nodige gegevens door te sturen naar de applicatie. De gegevens die wij gebruiken in de backend krijgen wij via Info Support. Info Support krijgt hun informatie weer van de NS. Onze app zal uitsluitend op tablets gebruikt worden dus er hoeft geen rekening gehouden te worden met telefoons of desktop. Alles binnen de ononderbroken lijn is digitaal en alles binnen de stippellijn zal door ons worden ontwikkeld middels de talen en frameworks binnen de stippellijn, voor de frontend is dit Angular Ionic met Cordova en voor de backend C# dotnet core.

2.2 Omgeving

Onze applicatie dient als proof of concept en staat los van het TimTim systeem, onze applicatie is daarom gericht om als lokale applicatie te draaien. In de daadwerkelijke implementatie van ons systeem zal integratie met TimTim worden gebruikt. Onze applicatie werkt door middel van een front- en backend systeem. Hiervoor is gekozen vanwege de grote hoeveelheid data en omdat de applicatie op een tablet moet draaien. De backend zal vooral verantwoordelijk zijn voor het verwerken van data om zo deze taak van de tablet te verlichten.

Er is de eis om de applicatie op verschillende tablets te kunnen draaien namelijk Android en iOS tablets. Om dit te verwezenlijken hebben we gekozen om de frontend te ontwikkelen in Ionic samen met Cordova en het Angular framework. Hierdoor kan men makkelijk onze applicatie op beide platformen draaien. Ook biedt deze keuze het voordeel dat op beide platformen de frontend hetzelfde uiterlijk heeft.

De backend is ontwikkeld in C# .Net core en ASP Core. De keuze voor C# kwam vanaf product owner aangezien hun systemen ook met C# werken. Wij hebben voor de C# .Net core variant gekozen vanwege het feit dat ons ontwikkelteam verschillende besturingssystemen gebruikt, hiernaast is .Net Core de meest recente versie van .NET en werkte het perfect voor onze doeleinden. Op de backend wordt de data van TimTim door middel van CSV bestanden in een database gezet er is dus geen externe connectie met TimTim.

Voor onze applicatie is het de bedoeling dat deze lokaal draait. Tijdens het ontwikkelen is er gekeken naar het opzetten van een centrale database maar omdat onze applicatie dient als proof of concept en omdat in het volledige systeem TimTim de database connectie voorziet hebben we samen met de product owner besloten dit niet op te nemen in onze scope.

Het uitgangspunt van de applicatie is dat deze zal worden gehanteerd bij de al in gebruik zijnde tablets van NS. Dit is de Samsung Galaxy Tab S2 (T. van den Hof, persoonlijke communicatie, 16 november 2018).

2.3 Architectuur

We hebben onder andere gekozen voor een framework met Angular omdat het een architectuur gebaseerd op componenten is. Ieder component bevat zijn eigen html, css en typescript. Componenten zijn herbruikbaar en goed geïsoleerd. Een component kan makkelijk worden aangepast zonder andere componenten te beïnvloeden.

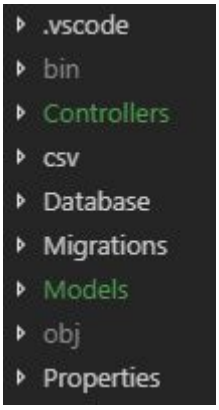
De applicatie moet worden gecompileerd met het ionic-serve commando. Hiermee wordt de applicatie voor een specifiek platform opgebouwd. Hierdoor werkt onze applicatie op IOS, Android en web browsers.

We hebben ervoor gekozen om een service-georiënteerde applicatie te maken omdat logica

in een service herbruikbaar is voor alle componenten in onze applicatie. Bovendien staat meer code op een centrale plek.

De integratie met van de front-end en de back-end wordt gedaan met de services en de api's. De services op de frontend roepen de api's van de back-end aan en halen zo de benodigde data op als json.

De backend dient alleen maar als api voor de frontend, dus we vonden een framework niet nodig. De backend heeft een simpele folder structuur:



Een voordeel van deze structuur is dat het simpel en overzichtelijk is.

Het is meteen te zien waar een bestand te vinden is. Onze Business Logic wordt alleen in de controllers geregeld zodat de database functies ook simpel en flexibel blijven.

We hebben tijdens het project geprobeerd om zo veel mogelijk grote functies op te delen in kleinere functies. Opgesplitst geven ze meer overzicht over de code en ze maken het schrijven van unit-tests makkelijker. Het is erg belangrijk dat code niet complexer is dan nodig, dit maakt het lastiger om code aan te passen en/of uit te breiden.

2.4 Testen

Om te testen worden testscenario's opgezet die de acceptatiecriteria goed encapsuleren, deze worden gedocumenteerd in het functioneel ontwerp. De uitvoeringen van deze testen zijn terug te vinden in de testplannen.

Voor de bewerkingen in de backend van het programma worden (indien meerwaarde aanwezig is) unit tests gemaakt. Deze unit tests zorgen ervoor dat de functies zo waterdicht mogelijk worden geprogrammeerd om problemen te voorkomen.

Wij zullen binnen onze applicatie geen End-to-End tests toepassen, een End-to-End test betekent het doorlopen van het programma op de frontend om te testen of de flow van de applicatie correct van begin tot eind verloopt.
Hier hebben wij voor gekozen omdat er binnen onze applicatie weinig acties plaatsvinden die een flow definiëren. Het is voornamelijk een statische pagina die informatie aan machinisten toont waar geen tot weinig acties van toepassing zijn.

3 Systeemontwerp

3.1 Ontwerpmethoden en standaarden

Voordat er begonnen wordt met programmeren moet er duidelijk zijn wat er geprogrammeerd moet worden, dit wordt gedaan door voor het programmeren een userstory te maken. In een userstory moeten een aantal dingen beschreven worden zoals: een omschrijving over de story, acceptatiecriteria, testscenario's en een ruw schermontwerp. Wanneer dit niet van toepassing is (backend) mag er begonnen worden met programmeren zodra er duidelijk is wat de acceptatiecriteria zijn, zodat de programmeur daar naar toe kan werken.

Wij maken gebruik van een Gitflow waar wij verschillende branches gebruiken om in te werken. Een Gitflow is een workflow dat de boomstructuur van een applicatie zo definieert dat je gemakkelijk een release kan pushen. Het is een goed raamwerk bij voornamelijk grotere projecten.
De layout van onze Gitflow is als volgt: er is een Master branch die als top dient, hier zit alle content in die in praktijk gebruikt kan worden. Een development branch waarin alle user stories die klaar zijn naartoe worden gepusht, deze user stories worden uitgewerkt in feature branches waarin elke feature wordt uitgewerkt.
Ook hebben wij binnen onze gitflow de mogelijkheid om hotfix branches aan te maken indien er incorrecte of kapotte code naar de development branch is gepusht en gelijk gefikst moeten worden.

3.2 Documentatie standaarden

Documentatie zal in het Engels geschreven zijn.

Er wordt zoveel mogelijk gebruik gemaakt van standaard C++ commentaar(Google 2018).

```
// Function to do a thing
// Calls another function to do the thing
// Returns specific value
```

De volgorde van imports binnen een module is ad hoc ontstaan; latere toevoegingen zijn later geïmporteerd. Binnen een component zelf wordt deze volgorde aangehouden:

```
// Angular

// Services

// Dependencies

// Components
```

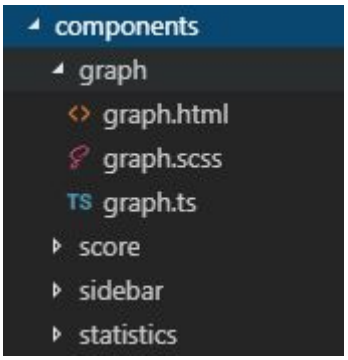
Wanneer code niet in één oogopslag duidelijk is, wordt deze voorzien van commentaar. Zeker als het niet uit de naamgeving of context duidelijk is.

```
        // Route to import rideprofiles. Imports the given filename
        given in the route
        [Route("rideprofile/{fileName}")]
        public List<RideNode> readCsvRideProfile(string fileName)
        {
            ...
        }
```

3.3 Programmeer standaarden

3.3.1 Projectstructuur

De structuur van onze frontend wordt grotendeels bepaald door Ionic en Angular. Het merendeel van onze structuur wordt dan ook automatisch gegenereerd. Hieronder is een voorbeeld van een automatisch gegenereerd component:



Bij onze backend houden we een MC-structuur aan. De backend wordt alleen gebruikt voor api calls, dus het heeft geen Views.
Naast de “Models” en “Controllers” hebben we ook nog een map genaamd “Csv.” Hierin kunnen alle CSV bestanden die uitgelezen moeten worden kunnen geplaatst.

3.3.2 Code standaarden

We hebben in dit project te maken met Javascript/Typescript voor onze frontend en C# voor onze backend.

Bij het schrijven van de backend code volgen we de officiële C# Code Conventions opgezet door Microsoft.

Voor het schrijven van de frontend code gebruiken we Typescript. Typescript is een superset van Javascript. Met Typescript kunnen we gebruik maken van typing, interfaces en betere classes. We volgen ook de volgende conventies:

- private variabelen beginnen altijd met een underscore: **private _router**.
- private variabelen worden niet gebruikt binnen de HTML.
- **const** wordt gebruikt voor een variabele als deze direct een waarde toegewezen krijgt die vervolgens niet meer veranderd wordt.
- **let** wordt gebruikt voor alle andere variabelen.
- Service imports krijgen de variabelenaam **“naam”**, en dus niet “**naamService”**
- Voor API calls wordt er gebuikt gemaakt van **async** en **await**.
- Variabelen en functies krijgen altijd een type mee. Voorbeeld:
  ```
  let message: string = "foo";
  ```

3.4 Software Development Tools

Voor het ontwikkelen van beide applicaties maken we gebruik van Team Foundation Server. Team Foundation Server biedt ons een centrale plek voor scrum boards, code, build systemen en automatische tests. Voor het werken in code gebruiken we Git versiebeheer en hanteren we de Git Flow werkwijze.
Om code te ontwikkelen hebben we gekozen om te werken in Visual Studio Code omdat dit werkt op meerdere operating systems en omdat we hier makkelijk zowel frontend en backend in kunnen ontwikkelen. Visual studio code werkt naast dit ook makkelijk samen met Team Foundation Server door de integratie in Visual Studio Code zelf.

Naast dit gebruiken we Android Studio om onze applicatie te kunnen testen en ontwikkelen op tablets. Android Studio biedt diverse tools om applicaties op het Android platform te kunnen testen/debuggen zoals emulators, processor en ram gebruik, en netwerkanalyse tools.

3.5 Openstaande issues

3.5.1 Verbeteringen

Een belangrijk punt wat helaas nog niet aanwezig is betere data om ideale ritten mee op te stellen. Hiermee kan er uitgebreider gekeken worden naar hoe een machinist hoort te rijden. Gedeeltelijk komt dit omdat TimTim nu vanuit de Tablet zelf data logt en andere loggingsystemen vallen op dit moment buiten de scope. Daarnaast worden externe gegevens niet ingevoegd, zoals bijvoorbeeld maximale snelheden of locaties van tunnels zoals bij Schiphol of Lelystad.

In de grafiek worden op dit moment nog geen duidelijke landmarks aangegeven. Een wens van NS en Info Support was om in ieder geval stations mee te geven. Dit is al mogelijk, maar was niet tijdig te realiseren.

De accuraatheid van sommige functies kan verder worden doorontwikkeld om tot betere resultaten te komen. Dit komt samen met het advies dat we gegeven hebben tijdens de eindpresentatie bij NS.

Er kan machinisten nog om feedback gevraagd worden om hun mening en gebruikerservaringen om verdere verbeteringen te ontdekken.

3.5.2 Open punten

De volgende punten komen niet af welke wel gewenst waren:

- Badges verdienen
- Ritten met bijbehorende gegevens vergelijken
- Verdere gamification

3.5.3 Toekomstige punten

De volgende punten zijn besproken en zijn wenselijk om later nog eens goed naar te kijken indien het project voortgezet zou worden:

- Verbeteringen van een machinist inzichtelijk maken
- Inzicht in ritten van meerdere machinisten
- Punten van ritten waar vaker vertraging voorkomt inzichtelijk maken
- Punten waar de rolcurve niet of slecht aangehouden wordt bijhouden
- Rekening houden met omvang van stations

4 Frontend

In dit hoofdstuk volgen de componenten die nodig zijn om de applicatie te realiseren. Eerst een overzicht van de gehele structuur, waarna dieper ingegaan wordt op de individuele componenten.

Services zijn aan te roepen om ervoor te zorgen dat er op een uniforme manier om wordt gegaan met bijvoorbeeld het formatteren van gegevens of het maken van api calls.

4.1 Hoofdlijn

De applicatie dient overzichtelijk gestructureerd te zijn. Dit komt terug in de locatie van de classes. Deze zijn ondergebracht in verschillende folders. Elke class is voor zijn eigen gedeelte verantwoordelijk, maar kan indien nodig gebruik maken van andere classes.

Wanneer herhaaldelijk dezelfde functionaliteit gebruikt wordt in meerdere componenten dient dit ondergebracht te worden onder een service. Hierdoor blijft het programma uniform en ontstaat er geen redundante code. Hoe de componenten met elkaar samenwerken blijkt uit de individuele componenten.

4.2 Individuele componenten

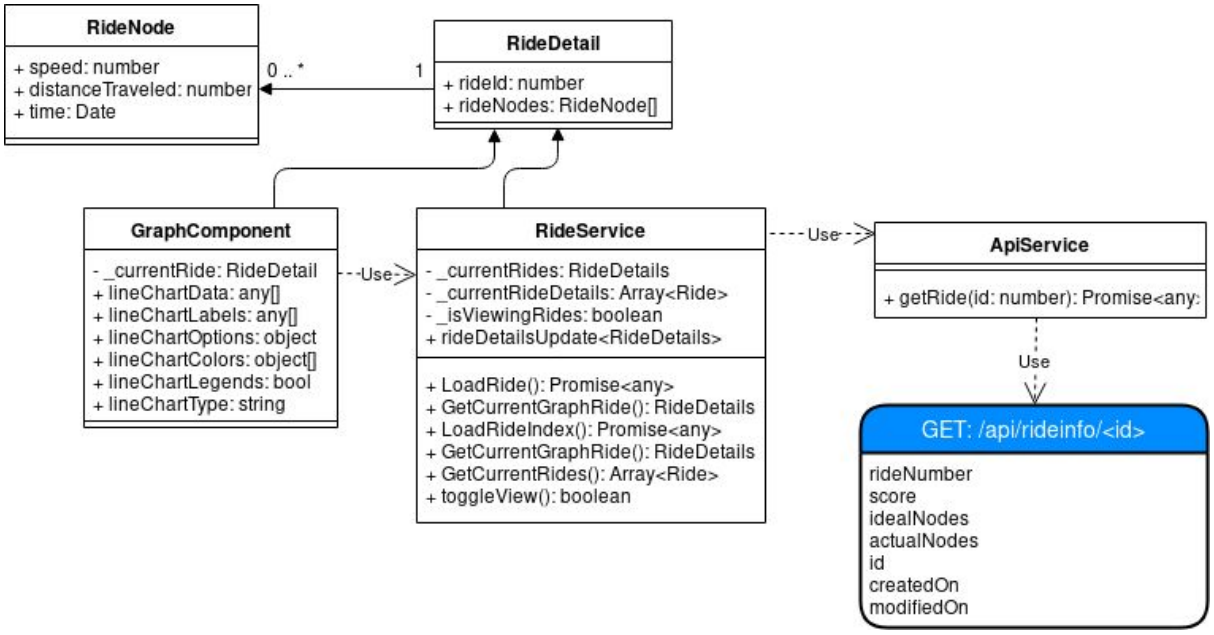
4.2.1 Graph

Userstories: RA-153, RA-339

Doel

Het grafiek component geeft de gereden rit in de frontend weer door de gelogde nodes in de grafiek te tonen met hierbij de tijd en afstand als x as en de snelheid als y as.

Diagram



Functies

Als de GraphComponent wordt geladen wordt de rit ingeladen door LoadRide() in RideService, deze voert de functie getRide() in ApiService uit die de api call regelt. Vervolgens kan deze rit worden opgehaald door de component door GetCurrentGraphRide().

Dependencies

- lodash
- ng2-charts
- chart.js

4.2.2 Advice

Doel

Het advies component laat advies zien van de gekozen rit. Default is hierbij de laatst gereden rit. Het advies omvat een korte titel en een beschrijving. Zo kunnen machinisten in een korte oogopslag zien waar het beter kan. De backend stuurt meerdere adviezen mee, maar enkel de top vier worden getoond.

Zo wordt er rekening mee gehouden dat er later indien nodig adviezen over specifieke ritdelen of doelen aanbevolen kunnen worden.

Advies 1	Titel	Beschrijving
Advies 2	Titel	Beschrijving
Advies 3	Titel	Beschrijving
Advies 4	Titel	Beschrijving
Advies 5	Titel	Beschrijving
Advies 6	Titel	Beschrijving
Advies 7	Titel	Beschrijving
Advies 8	Titel	Beschrijving
Advies 9	Titel	Beschrijving
Advies 10	Titel	Beschrijving
Advies 11	Titel	Beschrijving
Advies 12	Titel	Beschrijving
Advies 13	Titel	Beschrijving
Advies 14	Titel	Beschrijving
Advies 15	Titel	Beschrijving
Advies 16	Titel	Beschrijving
Advies 17	Titel	Beschrijving
Advies 18	Titel	Beschrijving
Advies 19	Titel	Beschrijving
Advies 20	Titel	Beschrijving

Functies

Met de LoadAdvice() functie wordt de api service aangeroepen via getAdvice(id), waarbij de id van de rit als parameter wordt meegegeven. Vervolgens kunnen de adviezen uit de advice service gehaald worden.

Dependencies

- lodash

4.2.3 Score

Doel

Het score component laat de behaalde score zien van de zojuist gereden rit. Deze score is opgedeeld in twee delen. De totale score wordt boven aan het component weergegeven, deze score wordt afgeleid van de behaalde deelscores. De deelscores worden weergegeven door middel van een subcomponent PartialScore. Deze componenten worden in het score component in een lijst weergegeven.

PartialScore bevat een icoon, titel en score. Door Middel van het icoon kan men snel zien hoe op deze deelscore is beoordeeld Duim omhoog voor goed, een informatie bol voor gemiddelde score of Duim omlaag voor slechte score.
Door het gebruik van een het sub component kunnen snel nieuwe deelscores aan de frontend worden toegevoegd mocht dit nodig zijn.

Functies

Als in het component op het info icoon wordt geklikt opent deze door middel van infoToggle functie de informatie box. Deze animatie box wordt met behulp van CSS animaties uit en ingeklapt.

De PartialScore componenten bepalen in hun constructors welk icoon getoond moet worden door middel van de CaculateIcon functie. Verder heeft dit component geen andere interactie mogelijkheden of animaties.

Dependencies

- FontAwesome

4.2.4 Statistics

Doel

Meer details tonen over de gereden rit, zoals topsnelheid en maximumsnelheid.

Functies

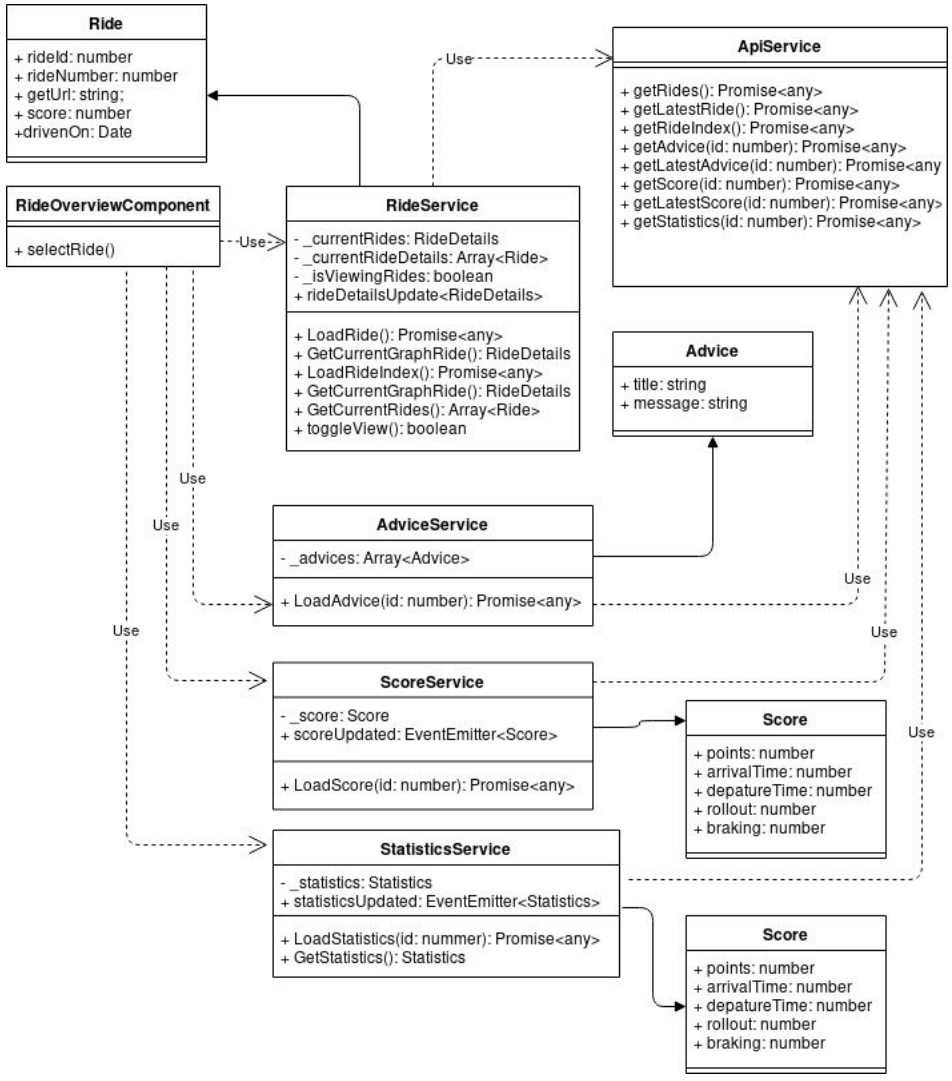
Statistics data word d.m.v. GetStatistics in services geladen.
Deze data wordt vervolgens weergeven in een tabel op de frontend.
De statistieken zitten in hetzelfde vak als “Score”. Om statistieken te zien moet er eerst op de “Statistics” knop gedrukt worden.

4.2.5 Ridehistory

Doel

Het doel van het ritgeschiedenis component is om machinisten inzicht te bieden op eerder gereden ritten. Het component zorgt na de selectie dat ook de andere componenten updaten met de data van de geselecteerde rit.

Diagram



Functies

De popup met de ritten is standaard gesloten wanneer de app laat, zodra in op de open oude ritten knop wordt geklikt wordt door middel van de toggleRideSelector() functie de popup geopend. Zodra de het RideOverview opent worden de oude ritten via de RideService opgehaald. Na het selecteren van een rit roept het component alle nodige functies aan in AdviceService, RideService, ScoreService en StatisticsService().

Dependencies

- RideService
- AdviceService
- ScoreService
- StatisticsService

5 Backend

5.1 Ontwerp Backend

Onze backend is ontwikkeld in .Net Core en maakt gebruik van ASP.NET Core. Hier is voor gekozen zodat de code op diverse platformen kan worden gedeployed (Windows, Mac en Linux).
.Net Core is de nieuwe versie van .Net en is aanzienlijk verbeterd in snelheid en volgens Microsoft bedoeld voor ‘High performance and scalable applications’. Naast dit is deze applicatie een losstaande proof of concept en zijn we niet afhankelijk van bestaande plugins die alleen in het .Net Framework beschikbaar zijn.

De backend werkt samen met de frontend door middel van een API. De API biedt de functionaliteit van backend aan zodat deze via de frontend gebruikt kan worden dit verlicht de processorkracht van de app. Naast dit voorziet de backend ook de database behoefte van de frontend, deze data is namelijk te groot om op de tablet te kunnen opslaan.

5.2 Structuur Backend

De backend is een C# applicatie ontwikkeld in .NET core en gebruikt ASP Core.
In de backend wordt data door controllers uitgelezen en in models geplaatst. Deze data wordt vervolgens door de frontend opgehaald via api-calls.

De backend bestaat naast de C# applicatie uit de database, deze database wordt opgemaakt door middel van het Entity Framework. Deze C# plugin helpt ons om de models makkelijk op te slaan in de database en werkt door middel van migraties. Dankzij migraties kunnen we makkelijk gezamenlijk aan de database te werken.
Omdat de backend geen visualisatie nodig heeft zijn er geen views aanwezig.

5.3 Controllers

5.3.1 CsvController

Om CSV bestanden te verwerken gebruiken we CsvHelper, CsvHelper is een open-source library voor C# en biedt uitgebreide functies om CSV bestanden te verwerken. CsvController leest de diverse CSV bestanden in en zet de data hierin om naar de juiste objecten in de backend. Met deze objecten kunnen verdere bewerkingen worden uitgevoerd, zoals score berekeningen of database operaties.

Sommige getallen in het CSV bestand gebruiken een komma als scheidingsteken en andere getallen gebruiken een punt. Als een getal komma's gebruikt dan moeten wij deze eerst omzetten naar punten om correct de data te verwerken.

CsvController verwerkt logging en ritprofiel CSV bestanden op verschillende wijze, dit omdat deze bestanden verschillende data bevatten. Alle regels in deze bestanden worden omgezet naar nodes. Logbestanden resulteren in een LogRideNode lijst en ritprofiel bestanden in een IdealRideNode lijst. Beide node types worden gebruikt in het berekenen van score, adviezen en visualisatie in de app.

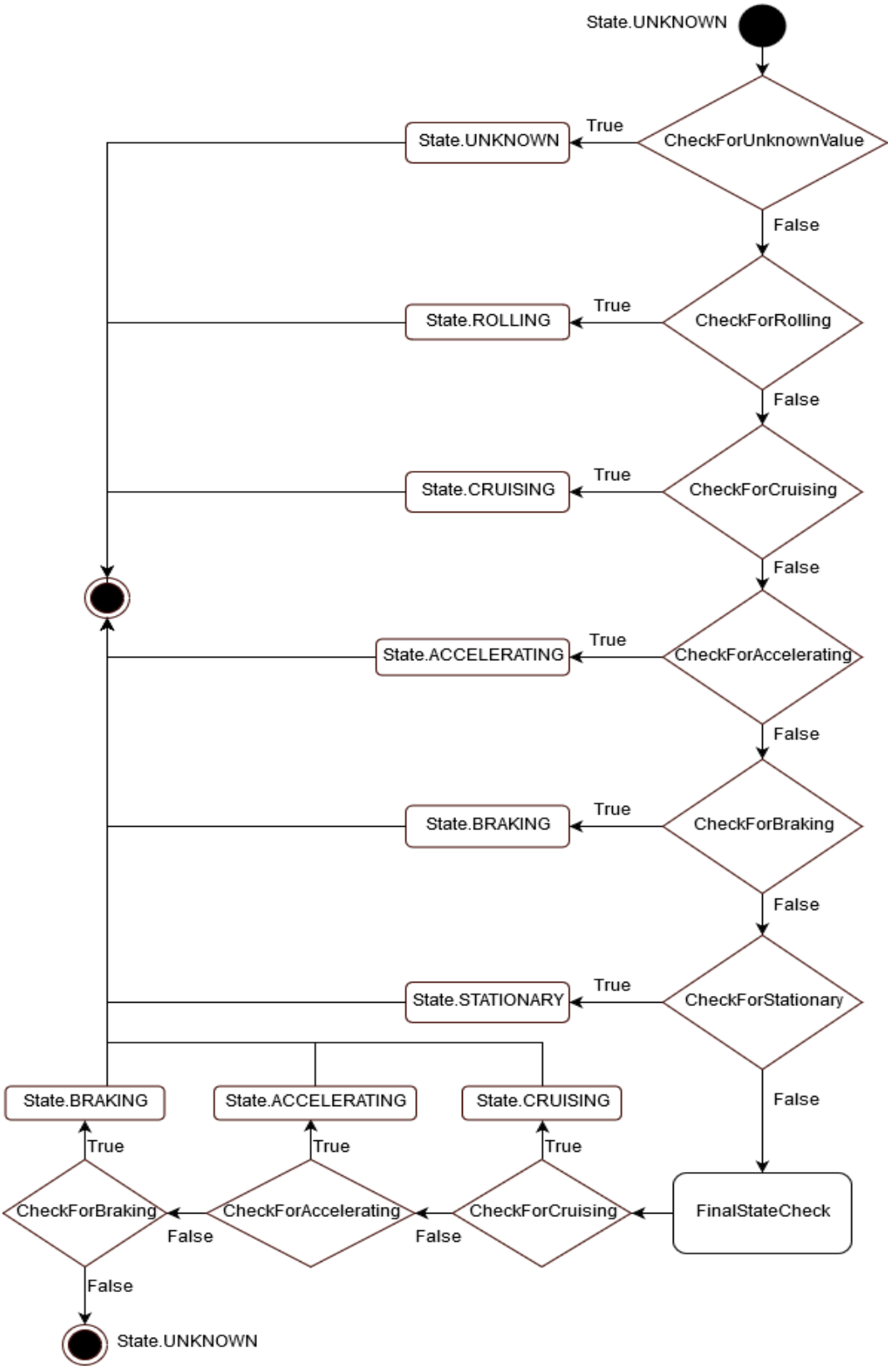
5.3.2 TrainStateController

De TrainStateController krijgt een route object binnen die omgezet is uit het CSV bestand. In een route wordt de staat van de trein bepaald door naar clusters van 5 nodes te kijken en door middel van een algoritme hier controle op uit te voeren.
Het algoritme kijkt naar de snelheidswaarde en de uitroltijd van de nodes en zal een enumerable waarde toekennen op basis van deze variabelen. De controller heeft nu een lijst met nodes waarin staat op welk moment een trein bijvoorbeeld aan het uitrollen is.

Het algoritme om de staat van de trein te bepalen werkt zoals eerder genoemd door naar het gemiddelde van 5 LogRideNodes te kijken die uit een logging bestand komen. Er is voor het gemiddelde van 5 LogRideNodes gekozen om tot een nog meer accuraat getal uit te komen in een NodeState object.. Dit algoritme kijkt naar de gemiddelde snelheid of het gemiddelde tijdsverschil van 3 NodeState objecten en zal op basis van deze waardes kijken(vergelijken) in welke staat de trein zich bevindt. Er zijn in totaal zeven checks die worden uitgevoerd om dit te bepalen, de checks zijn als volgt:

- CheckForUnknownValue: In deze functie wordt gekeken of er binnen de vijf LogRideNodes een waarde is de niet bruikbaar is, namelijk -1. Als hier sprake van is zal de staat van deze node “UNKNOWN” zijn.
- CheckForRolling: In deze functie wordt gekeken of er binnen de LogRideNodes het tijdsverschil van de nodes binnen een bepaalde marge gelijk blijft, dit betekent dat de trein aan het uitrollen is. Ook wordt er gekeken of de ritstatus wel op 4 staat, dit betekent dat de trein rollend het station zou kunnen bereiken. De staat van deze node zal dan “ROLLING” zijn.
- CheckForCruising: In deze functie wordt gekeken of er binnen de 3 Nodes en zal kijken of de gemiddelde snelheid van de nodes enigszins gelijk blijft, hier kan de conclusie uit getrokken worden dat de trein de staat “CRUISING” krijgt.
- CheckForAccelerating: In deze functie wordt er in de nodes gekeken of de snelheid van de nodes steeds verder omhoog gaat, binnen een bepaalde marge. De staat van deze node zal dan “ACCELERATING” zijn.
- CheckForBraking: In deze functie wordt gekeken of de gemiddelde snelheid van de nodes steeds verlaagt, hieruit kan de conclusie worden getrokken dat de trein zich in de staat “BRAKING” bevindt. Dit indien de ritstatus niet 4 is, sinds je hier eventueel ook kan zeggen dat trein aan het rollen is.
- CheckForStationary: In deze functie wordt gekeken of de snelheid van de nodes 0 is, indien dit waar is wordt de staat “STATIONARY” toegekend.
- FinalStateCheck: Deze functie dient als een laatste check indien wij via 3 nodes niet konden bepalen in welke staat de trein zat. Hier worden 2 nodes vergeleken(de eerste en laatste van de 3 originele nodes), deze nodes worden weer vergeleken om te kijken of de trein aan het accelereren, remmen of cruisen was. Als uit deze data nog steeds niets gehaald kan worden, trekken wij de conclusie dat de data van deze nodes niet bruikbaar is. Doordat deze functie twee nodes met elkaar vergelijkt is de data het niet geheel accuraat, maar zal indien de data goed gelogd wordt niet gebruikt worden.

In de onderstaande activity diagram is te zien hoe het algoritme doorlopen wordt.



5.3.3 StatisticsController

Haalt verschillende gegevens op en doet dit ook voor alle gegevens op een andere manier. Deze gegevens worden uit een lijst met LogRideNodes gehaald.
De volgende gegevens worden opgehaald:

TopSpeed: Pakt de hoogst gemeten snelheid uit de lijst met nodes te pakken.
AverageSpeed: Pakt de gemiddelde van alle snelheden uit de lijst met nodes te halen.
RolloutDistance: Wordt berekend met behulp van de TrainStateController berekend. Dit wordt gedaan door de afstand van alle nodes met de “ROLLING” status bij elkaar op te tellen.
IdealRolloutDistance: Wordt uitgelezen via de ritstatus. De tussen alle nodes waar ritstatus op vier staat wordt bij elkaar opgeteld.
RolloutRate: Wordt gekregen door simpelweg Uitrol Afstand door de Ideale Uitrol Afstand te delen.

5.3.4 AdviceController

De advicecontroller genereert advies objecten op basis van een gereden rid. Voor het optrek advies worden clusters van drie nodestates gecontroleerd op currentstate, als de eerste en de laatste accelerating zijn en de tweede niet dan kan er beter opgetrokken worden.
Voor het uitrol advies wordt er in de logridenodes gekeken naar het ideale uitrol moment en wanneer er daadwerkelijk begonnen is met rollen, als dit niet gelijk is wordt er een advies gegenereerd. Dit gaat doormiddel van de state die ontvangen wordt uit de trainstatecontroller. Voor speeding wordt gekeken naar de logridenodes of er harder is gereden dan het snelheidslimiet.

5.3.4 ScoreController

Een score wordt per rit gegeven aan de hand van hoe deze verlopen is.
Als vergelijkingsbasis dient de bijbehorende ideale rit.
Op vier punten wordt een score gegeven, welke tot een gewogen gemiddelde leiden.

Vertrek- en aankomstscore tellen dubbel, aangezien dit de belangrijkste punten van aandacht zijn voor de machinist om aan te houden. Uitrol- en remscore tellen enkel.

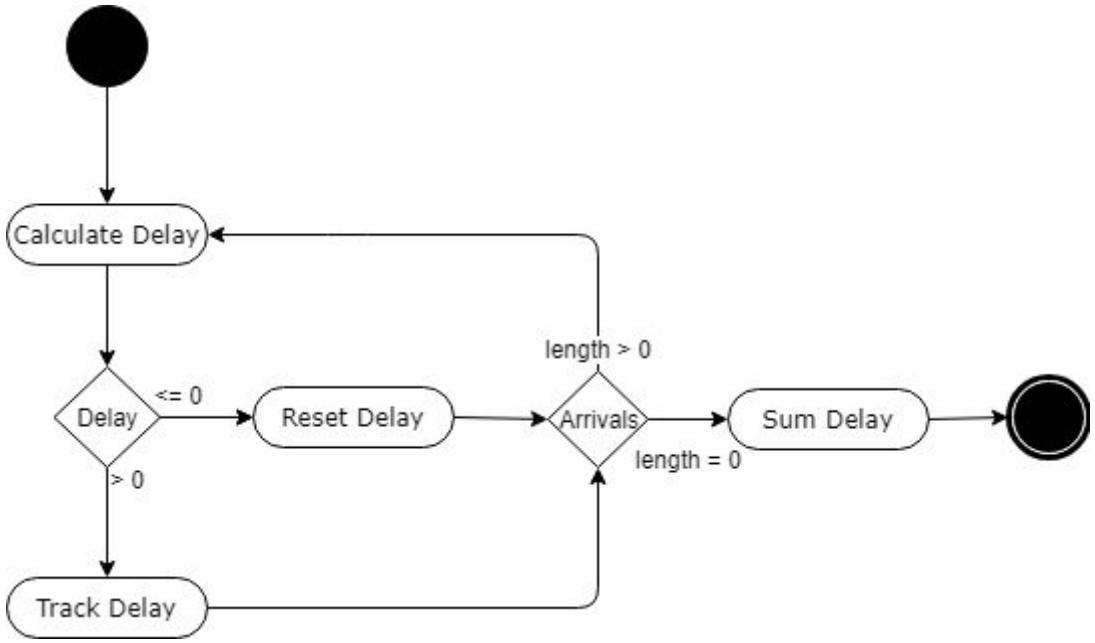
Scores zijn slechts een indicator om gemakkelijk een beoordeling van een rit te zien. Hierdoor is er in overleg gekozen voor een waardering tussen de 0 en 100.

Aankomstscore

Voor het bepalen van de score voor aankomsten wordt gekeken wanneer op bepaalde nodes aangekomen moet worden. Er is een marge waarmee gekeken wordt wanneer deze daadwerkelijk bereikt worden. De eerste node die hieraan voldoet wordt opgeslagen om met de ideale node te vergelijken.

```
LogRideNode arrivalNode = ActualNodes.First(x =>
(x.Coordinates.X < arrival.Item2.X + this._accuray && x.Coordinates.X > arrival.Item2.X - this._accuray) &&
 x.Coordinates.Y < arrival.Item2.Y + this._accuray && x.Coordinates.Y > arrival.Item2.Y - this._accuray);
```

Indien er vertraging is wordt deze opgenomen. Als hij te vroeg is dient er niets te gebeuren. Tijd inhalen wordt niet beloond. Hierdoor ontstaat de volgende situatie:



Een belangrijk aspect hierbij is dat doordat vertragingen bijgehouden worden een machinist nooit meerdere malen voor al eerder opgelopen vertraging wordt gestraft. Een kanttekening hierbij is dat wanneer vertraging ingehaald wordt de delay gereset wordt zodat verdere delays wel weer meegenomen worden. In een realistische situatie zou dit er dan zo uit zien:

Actual	Ideal	Delay	Tracked Delay	Calculated Delay	Sum
12:00	12:00	0	0	0	0
12:15	12:13	2	2	2	2
12:20	12:18	2	0	0	2
12:26	12:27	-1	0	0	2
12:35	12:32	3	3	3	5
12:44	12:42	2	2	0	5

Dus alhoewel er meerdere keren vertragingen zijn geweest tellen ze alleen mee indien er geen of kleinere vertragingen vooraf gingen.

Vertrekscore

Vertrekscores worden op een vergelijkbare manier als aankomstscore berekend. Nu worden alleen vertreknodes vergeleken. Er wordt nu gekeken naar welke laatste actuele node voldoet aan de locatie van de ideale node.

Remscore

Vanuit de TrainState wordt bijgehouden hoeveel remmomenten er zouden moeten zijn en hoeveel er daadwerkelijk geweest zijn. Dit wordt naar een score omgezet.

Rolscore

De ideale hoeveelheid dat een trein uitrolt en de daadwerkelijke hoeveelheid uitrollen worden uit de RideStatisticsController gehaald en naar een score omgezet.

5.4 Models

BaseModel

BaseModel is een class die door veel models wordt gebruikt als base class. Het bevat twee DateTime velden: CreatedOn en ModifiedOn. Wanneer een class wordt opgeslagen welke overerft van BaseModel vult de database deze waardes automatisch in (zie Database 5.6). Deze waarden kunnen in de rest van de applicatie worden gebruikt en helpen bij het organiseren en structureren van de data in de database.

- Id: Het idee van het model in de database.
- CreatedOn: Datum wanneer het model voor het eerst is aangemaakt in de database.
- ModifiedOn: Datum van de laatste modificatie van het model.

LogRideNode / IdealRideNode

Nodes bevatten informatie uit de CSV bestanden, voor elke regel in een CSV bestand wordt een Node aangemaakt. IdealRideNode bevat de data uit een ritprofiel, LogRideNode bevat de data van een logging CSV bestand.

LogRideNodes bevat de volgende waardes:

- Time: Tijd van de meeting
- RideStatus: De status van de rit op dit punt.
- TimeTableCode: Dienstregelpuntcode.
- DistanceToStation: Afstand tot het station.
- TimeDifference: Tijdverschil op het moment van de meeting.
- Speed: Snelheid op het gemeten punt
- MeasuredSpeed: Gemeten Snelheid
- Coordinates: Rijksdriehoekscoördinaten van de trein.

IdealRideNodes bevat de volgende waardes:

- TimeTableCode: Dienstregelpuntcode.
- ArrivalTime: Tijd van aankomst.
- DepatureTime: Tijd van vertrek.
- ActionType: Actietype van de trein (Aankomst, Vertrek, Doorgang, Onbekend).
- Coordinates: Rijksdriehoekscoördinaten van de trein.
- DistanceToStaton: Afstand tot het station.

RideInfo

RideInfo objecten bevat de lijst van IdealNodes en ActualNodes. Deze data wordt gebruikt om diverse andere aspecten van de rit te bereken. Naast dit wordt er in de RideInfo ook andere handige informatie van rit opgeslagen, Alle waardes in een RideInfo object zijn:

- RideNumber: Ritnummer van de rit.
- Score: Berekende score van de rit.
- IdealNodes: Lijst met Ideale nodes waarmee scores en andere waardes berekend.
- ActualNodes: Lijst van alle LogNodes van de gereden rit.
- DrivenOn: Datum waarop de rit gereden is.

ConvertedRideInfo

Dit is een extensie van RideInfo. Een ConvertedRideInfo object bevat nog een extra property. Deze property wordt niet uit de database gehaald maar wordt gegenereerd door de NodeConversionController, dus we vonden dit niet thuishoren in RideInfo. Dit model heeft de volgende properties:

- IdealNodesAsLogNodes: Een lijst met Lognodes die door NodeConversionController zijn gegenereerd uit de lijst met IdealNodes.
- RideNumber: Ritnummer van de rit.
- Score: Berekende score van de rit.
- IdealNodes: Lijst met Ideale nodes waarmee scores en andere waardes berekend.
- ActualNodes: Lijst van alle LogNodes van de gereden rit.
- DrivenOn: Datum waarop de rit gereden is.

RDCoordinates

Kleine class die de rijksdriehoekscoördinaten representeert samen met de nauwkeurigheid van deze coördinaten. Bij beide nodes wordt deze class gebruikt om data coördinaten weer te geven.

- X: X as van het coördinaat.
- Y: Y as van het coördinaat.
- Accuracy: De nauwkeurigheid van het coördinaat..

TrainState / NodeState

Bevat een lijst met NodeStates.
De NodeStates zijn clusters van vijf LogRideNodes die een status toegewezen hebben gekregen door middel van de TrainStateController. Deze status wordt opgeslagen middels een enumerator die verschillende waardes kan hebben. Deze waardes zijn:

- Onbekend
- Optrekken
- Remmen
- Rollen
- Stilstaan
- Cruisen

Tevens worden de gemiddelde snelheid en de gemiddelde uitroltijd opgeslagen in de NodeState.
Deze models bevatten de volgende waardes:

- Nodes (TrainState): Een lijst met nodes waarin de staat van de trein is bepaald.
- Nodes (NodeState): Een lijst met nodes waar de staat nog niet bekend is.
- CurrentState: De staat van een NodeState node.
- AverageSpeed: Het gemiddelde van vijf nodes van een NodeState node.
- AverageTimeDifference: Het tijdsverschil van vijf nodes van een NodeState node.

Advice
Advice objecten bestaan uit drie attributen: een titel, een bericht en een prioriteit. Dit object wordt frontend gebruikt om advies weer te geven. Omdat er in eerste instantie maar vier adviezen worden weergegeven wordt er een prioriteit meegegeven zodat de belangrijkste adviezen altijd bovenaan staan. Indien er meer dan vier adviezen getoont moeten worden kunnen ze ook makkelijk op prioriteit worden gesorteerd.

Statistics
In een Statistics object staan alle statistieken die bij een rit horen.
Statistics bevat de volgende waardes:

- MaxSpeed: De maximale snelheid die is gereden op basis van een rit log.
- AverageSpeed: De gemiddelde snelheid die is gereden op basis van een rit log.
- MaxRolloutDistance: De ideale uitrolsnelheid op basis van het ritprofiel.
- RealRolloutDistance: De uitrolsnelheid tijdens een rit op basis van de log.
- RolloutRate: Hoeveel van de maximale uitrolafstand ook daadwerkelijk is uitgerold.

Badges
Badges hebben 3 attributen.

- Title: De titel van de badge die frontend wordt weergegeven.
- Description: De uitleg bij een badge die wordt getoont als je badge details weergeeft.
- Image: Een identifier zodat de frontend weet welke afbeelding bij de badge getoont moet worden.

Driver
Driver bestaat alleen zodat we in de database badges aan een machinist kunnen koppelen. Op dit moment heeft een driver dus alleen een lijst met badges.

Score
Een Score object moet worden aangemaakt met verschillende attributen die samen een totaalscore vormen. De bedrijfslogica is hier niet in opgenomen en zal in de desbetreffende controller meegenomen worden. Een score bevat de volgende attributen:

- Points: Aantal punten na weging
- ArrivalTime: Aantal punten voor aankomst en doorkomsttijden
- DepartureTime: Aantal punten voor vertrektijden
- Rollout: Aantal punten voor rolduratie
- Braking: Aantal punten voor remmen

5.5 API

In dit hoofdstuk zal kort worden ingegaan op de API in het algemeen. De API dient in onze applicatie alleen als een methode om data op te halen, hiermee verlichten we de frontend van het uitvoeren van processor intensieve berekeningen en hiermee houden we de grote van de frontend applicatie ook beperkt aangezien onze dataset veel schijfruimte in beslag neemt.

De api wordt niet gebruikt om data van de gebruiker terug in de database te plaatsen omdat deze interacties niet voortkomen in onze applicaties. Alle data van de API wordt teruggestuurd als Json data, Json biedt ons de beste manier om de data makkelijk te kunnen integreren door de ingebouwde support voor Json data in Javascript (en Typescript).

De volledige lijst van endpoints beschikbaar in onze api is:

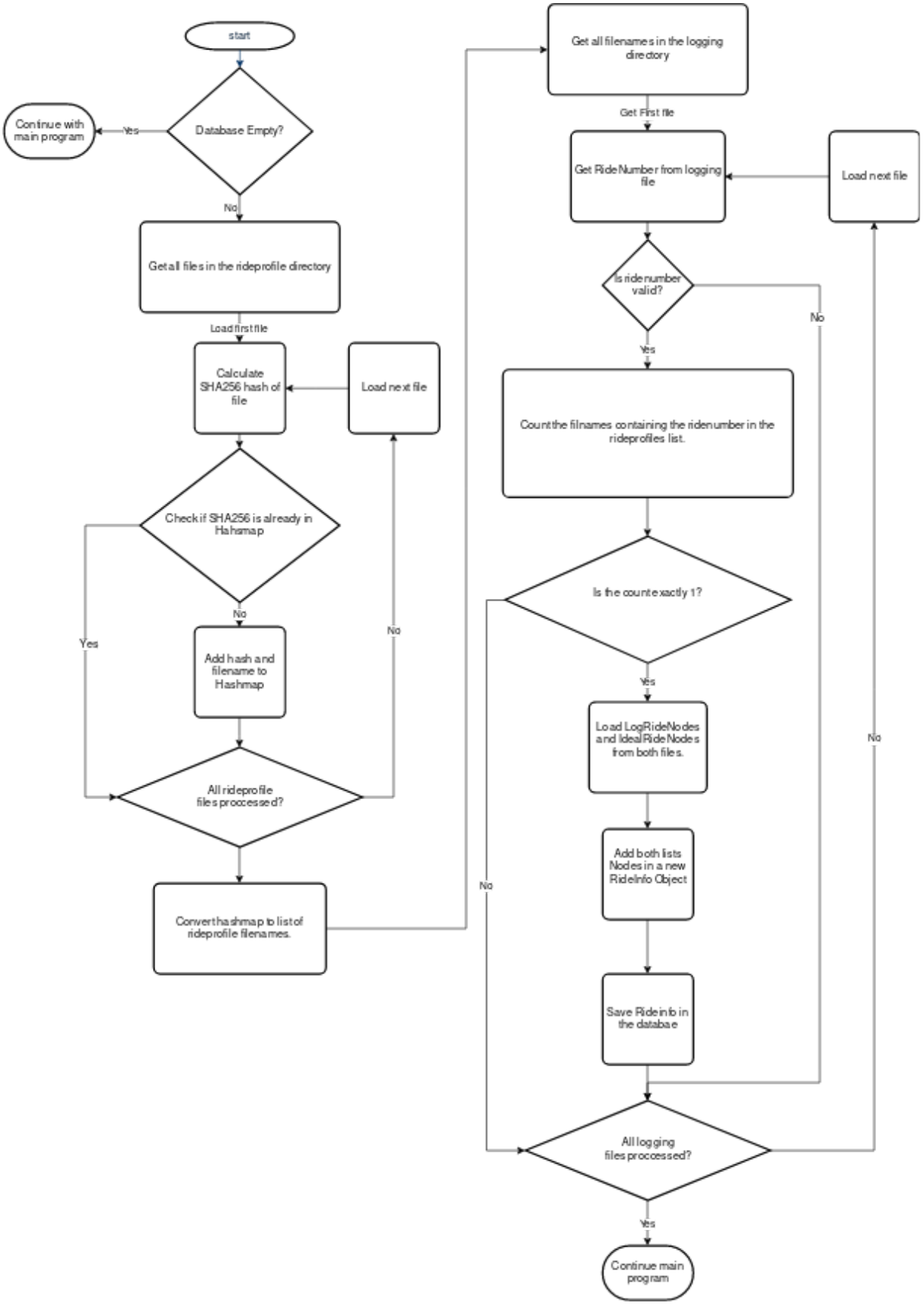
```
/api/rideinfo
     Geeft alle verkorte lijst van alle rideinfo objecten

/api/rideinfo/[ id | latest | oldest ]
     Geeft 1 specifieke ridenode
     of 404 als geen gevonden.

/api/advice/[ id | latest | oldest ]
     Geeft advies voor een ridenode met id weer
     of 404 wanneer niets gevonden.

/api/badge/<id>
     Geeft badges van een machinist met <id>
     of 404 wanneer niets gevonden.

/api/score/[ id | latest | oldest ]
     Geeft 1 score object van een specifieke rideinfo
     of 404 wanneer niet gevonden
     of 422 wanneer een rit niet verwerkt kan worden.

/api/ridestatistics/[ id | latest | oldest ]
     Geeft statistics weer van een rideinfo
     of 404 wanneer niet gevonden.
```

5.6 Database

Voor het vergelijken van scores en het bijhouden van badges is het nodig om een database voor onze applicatie bij te houden. Omdat deze database sneller werkt dan het inladen van CSV bestanden vanaf de harde schijf is ook gekozen om CSV bestanden in deze database op te slaan. Voor het opslaan van de CSV bestanden waren echter diverse stappen nodig om te zorgen dat alleen valide data in de database opgeslagen wordt.

In de collectie van CSV bestanden zijn namelijk diverse identieke bestanden te vinden, ook zijn er diverse verschillende ritprofielen met hetzelfde ritnummer en ontbreken bij enkele logbestanden het bijbehorende ritprofiel. Door de grote hoeveelheid CSV bestanden is er gekozen om bij het inladen van deze bestanden extra checks en validatie toe te voegen. Deze validatie zal ik in het dit hoofdstuk verder toelichten.

5.6.1 Flowchart



Bovenstaande flowchart voorkomt dat er meerdere ritprofielen worden ingeladen bij het zelfde ritnummer en zorgt dat alleen unieke bestanden worden ingelezen door de SHA-265 sum check. Mochten er meerdere ritprofielen met unieke inhoud in de dataset voorkomen met hetzelfde ritnummer worden deze ook niet ingeladen, Dit omdat uit de ritprofiel bestand niet opgemaakt kan worden welke nieuwer is en moet worden gebruikt.

Omdat we niet weten welke van de ritprofiel bestanden gekozen moet worden ingeval van dubbele bestanden hebben we nadrukkelijk besloten geen van de ritprofielen in te laden. Een verouderd ritprofiel kan namelijk leiden tot het makkelijker of moeilijker halen van scores om de gamification aspecten van het project zo eerlijk mogelijk te houden worden deze situaties daarom ook zoveel mogelijk vermeden.

5.6.2 Automatisering

Om de database te organiseren wordt gebruikgemaakt van CreatedOn en ModifiedOn waardes, deze waardes worden automatisch ingevuld wanneer een model een uitbreiding is van BaseModel. Hier is voor gekozen om de integriteit en organisatie van de Database te verhogen. Deze waardes kunnen gebruikt worden voor bijvoorbeeld het terugzetten van backups of het bijhouden wanneer een onverwachte wijziging is doorgevoerd in de database.
De CreatedOn waardes zijn opgenomen om de database te organiseren, ook kunnen met deze waardes queries worden uitgevoerd die beperken op aanmaakdatum waar dit voorheen niet standaard kon.

5.6.3 Database classes

Om de database te implementeren zijn enkele classes toegevoegd om makkelijker met de database te kunnen werken.

ApiContext

Database context voor de applicatie bevat alle DbSets van de applicatie en breidt de standaard database context uit met extra functionaliteit. Zo worden alle modellen die BasisModel uitbreiden standaard voorzien van een datum van aanmaken `CreatedOn` en bij elke update wordt de waarde `ModifiedOn` geupdate. Dit maakt het makkelijker de veranderingen in de database bij te houden en biedt mogelijkheid om de gegevens in de database makkelijker te sorteren.

- Audit: Deze functie wordt aangeroepen voor elke saveChanges om te zorgen dat default waardes CreatedOn or ModifiedOn worden gevuld.

DbContextExtenstion

Deze class is verantwoordelijk voor het inladen van de CSV bestanden in de database en checkt of alle migrations in zijn uitgevoerd voor het opstarten van de applicatie. Berekent ook de Sha256sum van de CSV bestanden om te voorkomen dat bestanden dubbel in de database worden ingeladen.

- AllMigrationsApplied: Returns true als alle migrations verwerkt zijn.
- GetFileHash: Berekent de Sha256sum van een bestand
- GetRideProfileFiles: Haalt een lijst met alle unieke rideprofile bestanden op.
- LoadingCSVFiles: Laad alle CSV bestanden die geladen kunnen worden en slaat deze op in de database als RideInfo objecten.
- EnsureSeeded: Deze functie wordt aangeroepen wanneer de applicatie gestart is en roept LoadingCSVFiles aan wanneer er geen RideInfo objecten in de database gevonden zijn.

5.6.3 ERD