Chiem Stevens

Cp.stevens@student.avans.nl

28-03-2021

Portfolio internettechnologie

Chiem Stevens

Inhoud

[Inleiding 2](#_Toc67937750)

[Tour of heroes 3](#_Toc67937751)

[Observables en subscribe 3](#_Toc67937752)

[RxJS operators 4](#_Toc67937753)

[pipe 4](#_Toc67937754)

[map 4](#_Toc67937755)

[find 4](#_Toc67937756)

[Subject 5](#_Toc67937757)

[Input/ output 5](#_Toc67937758)

[Verreken app 6](#_Toc67937759)

[Devextreme 6](#_Toc67937760)

[Design 6](#_Toc67937761)

[Code 9](#_Toc67937762)

[Models 10](#_Toc67937763)

[Algoritme 11](#_Toc67937764)

[Bijlage 12](#_Toc67937765)

[Bijlage 1: Algortime in code 12](#_Toc67937766)

[Bijlage 2: Algoritme als diagram 14](#_Toc67937767)

[Bibliografie 15](#_Toc67937768)

# Inleiding

Voor het vak internettechnologie maakt de student kennis met Angular. Voor mijn stage begin leerjaar 3 heb ik ook gewerkt met Angular, ik heb daarom wat voorkennis opgedaan. In de beoordeling van het vak staat dat er een duidelijk leercurve aanwezig moet zijn, dit is mijn geval wat moeilijk aan te tonen omdat ik al een goede basis heb opgebouwd met Angular. Ik hoop echter toch te laten zien dat ik voldoende kennis beschik over Angular met mijn uitleg over de Tour of Heroes en de verreken app, zodat ik dit vak toch met de juiste competenties op zak kan afronden.

# Tour of heroes

De tour of heroes is een tutorial gemaakt door Angular om het framework te leren begrijpen. Voor mijn stage begin leerjaar 3 heb ik ook gewerkt met Angular, hier heb ik ook de tour of heroes doorlopen. Ik heb voor het vak internettechnologie nogmaals de tutorial doorlopen, ik heb **NIET** alle stappen van de tutorial in dit hoofdstuk opgenomen, inplaats hiervan heb ik mij gefocust op de onderdelen die mij aanspreken en waar ik tijdens dit vak nog iets van heb geleerd. Deze onderdelen zijn hieronder terug te vinden.

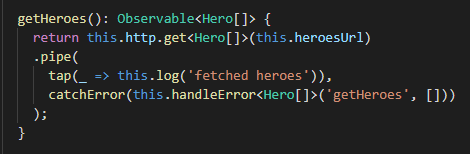
## Observables en subscribe

Een van de mooiste onderdelen van Angular vind ik de observables. Angular maakt gebruik van observables als een interface om een groot aantal veelvoorkomende asynchrone bewerkingen uit te voeren. In figuur 1 hieronder geeft de methode getHeroes een observable terug, er wordt hier op gesubscribeerd om de waarde die terugkomt op te slaan (in het voorbeeld hieronder gebeurt dat door de lokale variabele heroes te vullen met de heroes die terug worden gegeven).



Figuur : Voorbeeld van het subscriben op een observable.

Figuur 2 beschrijft wat de getHeroes methode teruggeeft, hier is onze observable ook in terug te vinden! De methode geeft aan dat er een Observable terug wordt gegeven van het type Hero array. In deze methode zelf wordt er een http get request gedaan, het resultaat hiervan wordt opgehaald door op de observable te subscriben. Verder zien we nog iets interessant in deze methode gebeuren, het gebruiken van de pipe operator, hier wordt in het volgende hoofdstuk meer uitleg over gegeven.



Figuur : De getHeroes methode die een observable terug geeft.

## RxJS operators

De observables zijn gaaf om te gebruiken, maar komen pas echt tot leven door het gebruik van RxJS operatoren. Observables zijn de basis, maar de operatoren zijn de essentiële onderdelen waarmee complexe asynchrone code eenvoudig op een declaratieve manier kan worden samengesteld (operators, 2021). In dit hoofdstuk worden een paar van deze operators behandeld die ik interessant vind om te gebruiken.

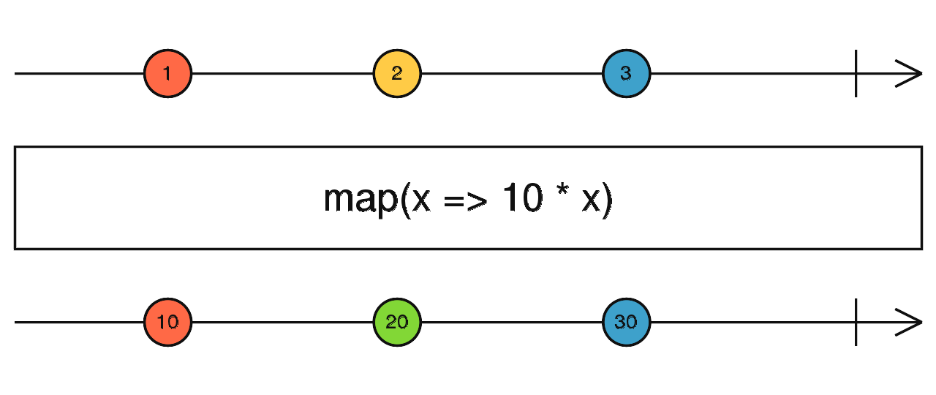
### pipe

Het komt in de praktijk vaak voor dat meerdere operatoren op een observable gebruikt dienen te worden. Observables hebben daarom een methode genaamd pipe, waarmee meerdere operatoren op een observable uitgevoerd kunnen worden. In code ziet dit er ongeveer zo uit:

obs.pipe( op1(), op2(), op3(), op3(), );

### map

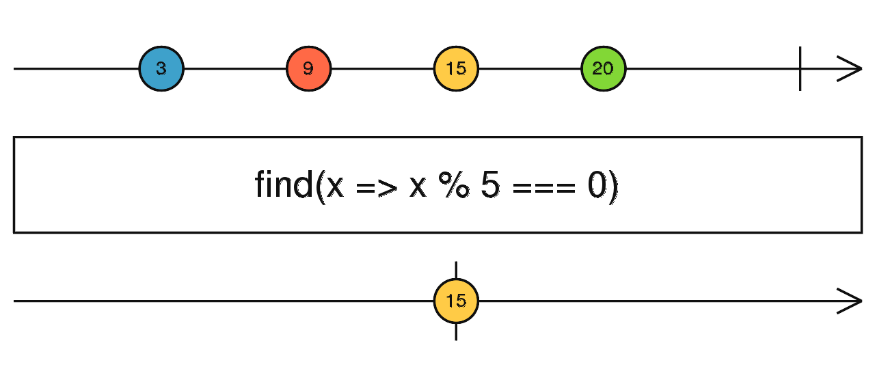
De map operator wordt gebruikt om elk resultaat door een transformatie functie te sturen, de waardes die hieruit komen worden opgeslagen in de observable.



Figuur : De werking van de map operator

### find

find zoekt naar het eerste element dat in de observable zijn elementen voorkomt, dat aan de meegegeven conditie voldoet. De find operator word in de verreken app gebruikt om personen met openstaande betalingen te zoeken.



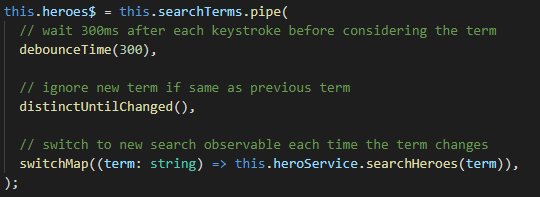
Figuur : de werking van de find operator

## Subject

Een subject is ook onderdeel van RxJS, een RxJS-subject is een speciaal type observable waarmee waarden kunnen worden gemulticast naar veel observers. Het komt er eigenlijk op neer dat er meerdere keren gesubscribed kan worden op een subject, waarbij dit maar een keer kan bij een observable (het kan wel bij een observable, maar het geeft alleen waardes terug aan een van de twee observers). In tour of heroes wordt een subject als volgt gebruikt:



Figuur : Een subject wordt op deze manier aangemaakt



Figuur : De RxJS operatoren aangeroepen in het subject

## Input/ output

Om variabele te delen tussen componenten in Angular word er gebruik gemaakt van de Input en Output annotaties. Hiermee kan de variabele gedeeld worden tussen parent en child componenten.



Figuur : Het aanmaken van een Input variabele

Om de variabele mee te geven aan de parent kan de volgende regel worden gebruikt.



Figuur : Het meegeven van een de Input variabele vanaf de parent

# Verreken app

Als eindopdracht voor het vak internettechnologien dient er een verreken app gemaakt te worden. Voor het maken van deze app heb ik gebruik gemaakt van devextreme. Meer hierover is in dit hoofdstuk nog te lezen. Op het design dat in de opdracht is gegeven is het mogelijk om een naam en een bedrag in te vullen. Ik heb ervoor gekozen om inplaats van een naam invullen een lijst met namen aan te leveren, hier heb ik voor gekozen omdat ik deze lijst uit de backend wou laten komen, echter is het niet meer gelukt om een backend te realiseren, deze data word nu uit een fake backend service gehaald.

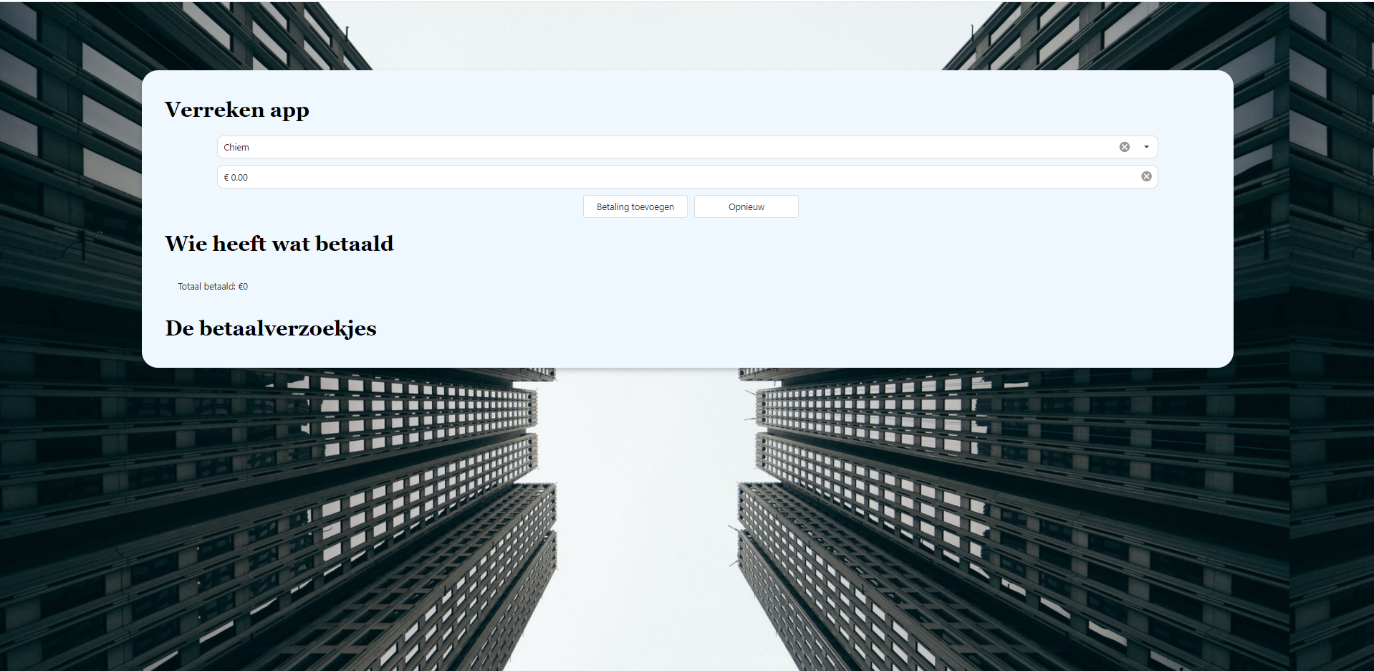
## Devextreme

Op mijn stage heb ik kennis gemaakt met devextreme. Devextreme is een library voor Angular om material design te versimpelen. Het word aangeleverd met veel guides en demo’s. Meer informatie over devextreme is te vinden op: <https://js.devexpress.com/>

In de verreken app heb ik devextreme gebruikt om de app er beter uit te laten zien, ook handelt het input validatie op een simpele manier af.

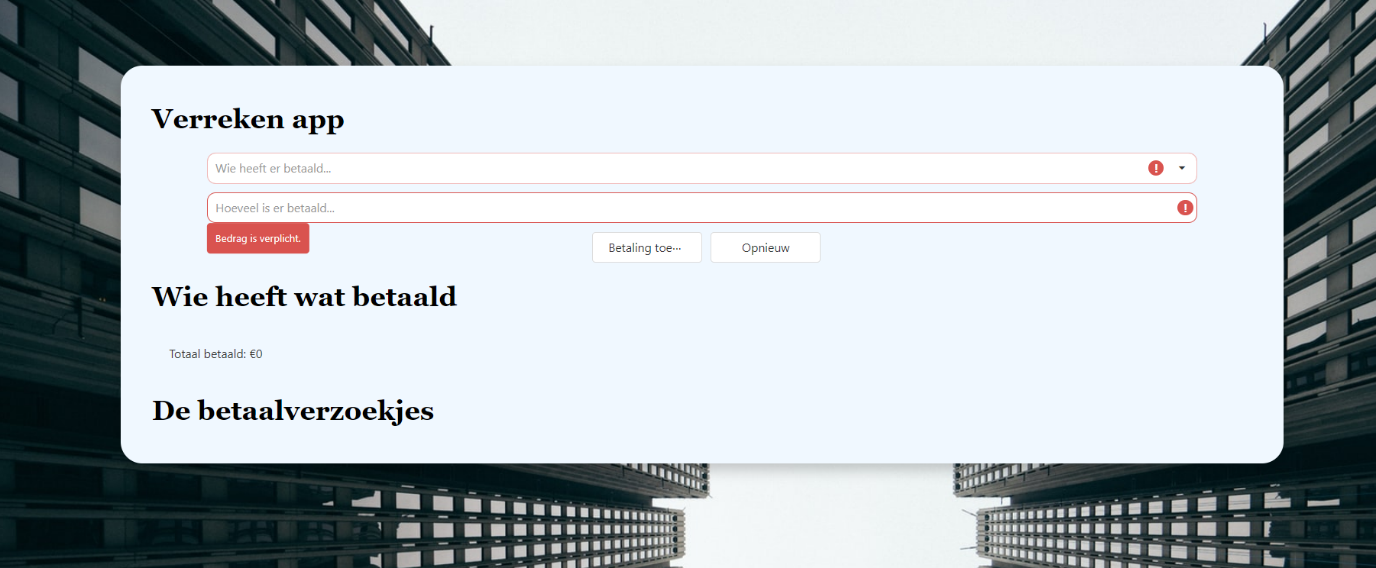
## Design

De app heeft twee invoervelden: de naam van de persoon die heeft betaald, en het bedrag dat die persoon heeft betaald. Daaronder zijn twee knoppen aanwezig, een om de betaling toe te voegen, en de ander om alle betaal verzoeken en ingevoerde waardes te resetten. Daaronder staat de wie heeft wat betaald, bij het invoeren van een betaalverzoek komt hier te staan wie heeft betaald, later is het mogelijk om hier ook aan toe te voegen waarvoor. Daaronder worden de betaalverzoekjes aangemaakt.



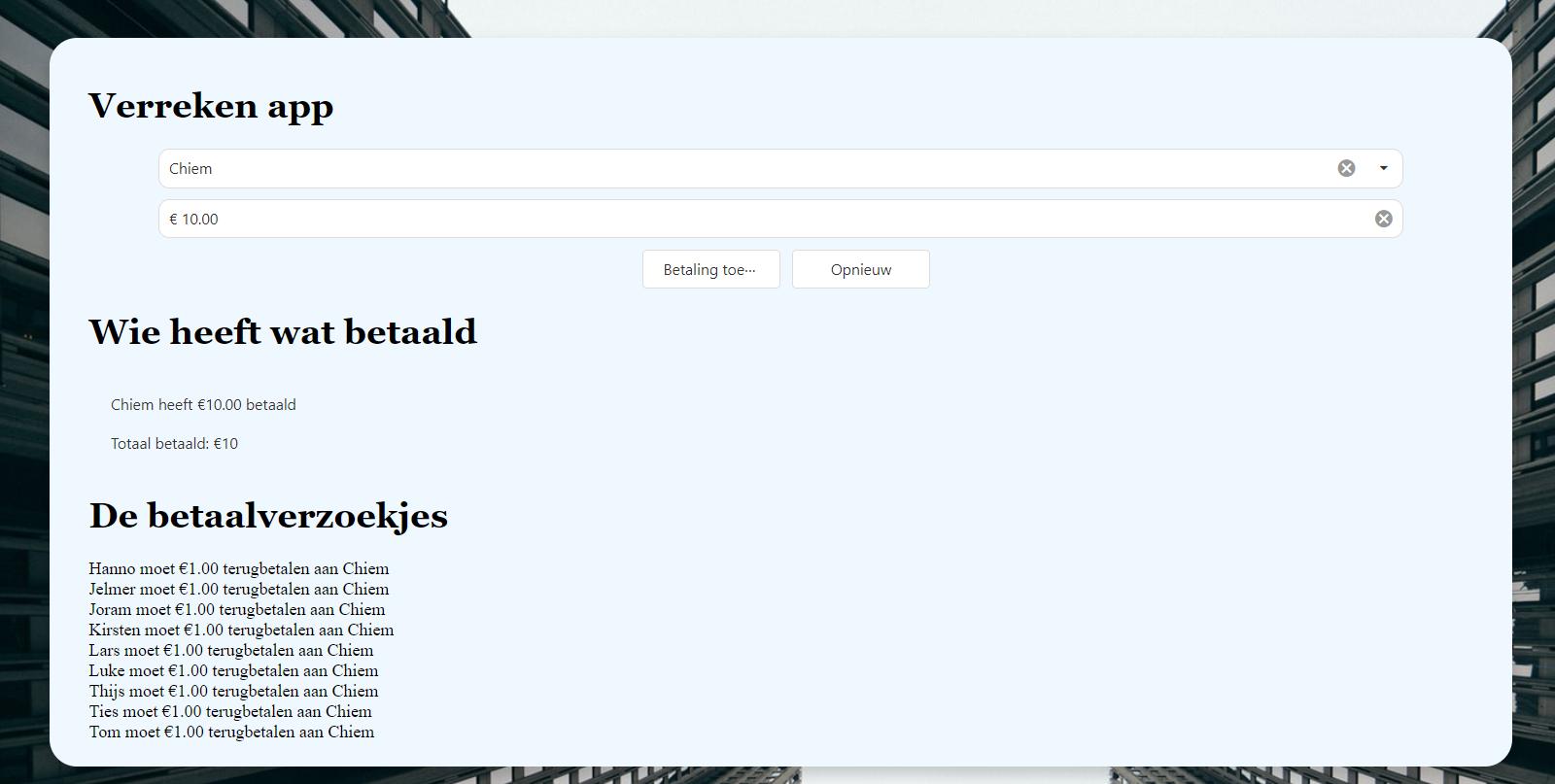
Figuur 9: Verreken app na het opstarten

Op beide invoervelden zit input validatie, zo mag de naam niet leeg zijn, en moet het bedrag een nummer zijn met maximaal twee decimalen.



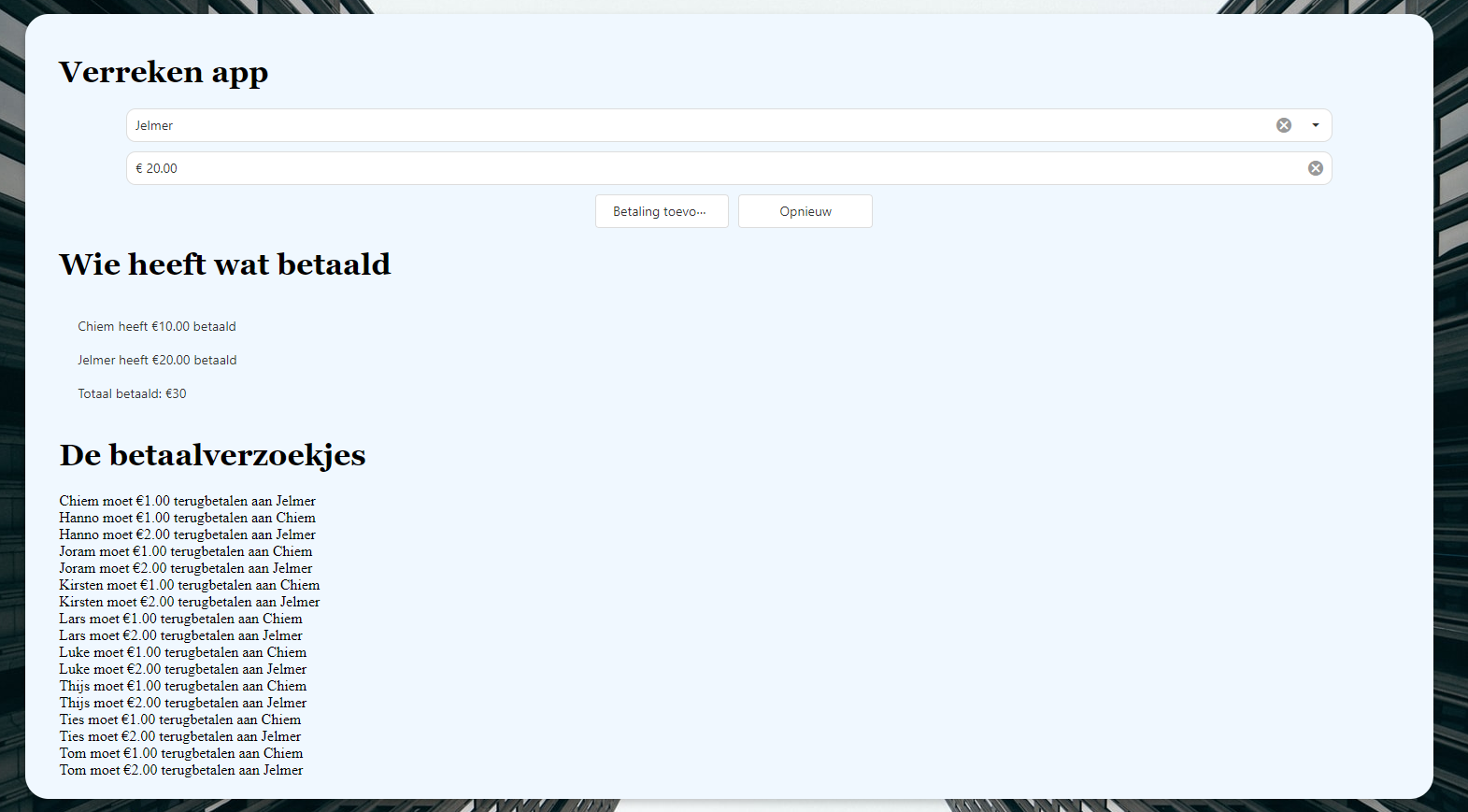
Figuur : Inputvalidatie aanwezig bij de invoervelden

Na het invoeren van een betaling worden de betaalverzoekjes aangemaakt.



Figuur : Verreken app met betaalverzoekjes

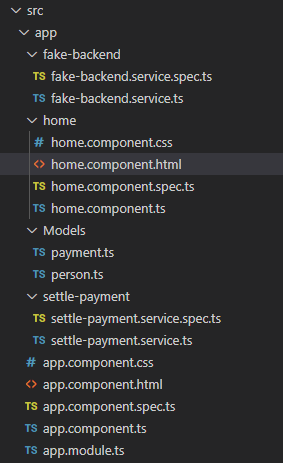
Het is ook mogelijk om meerdere betalingen door meerdere personen toe te voegen.



Figuur : Betaalverzoekjes bij betalingen door meerdere personen

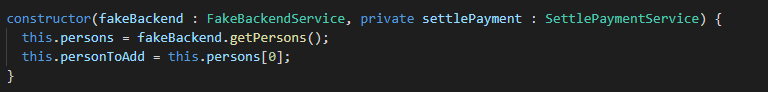
## Code

De structuur van de bestanden ziet er als volgt uit:



Figuur : Bestandenstructuur in verreken app project

De app bestaat maar uit een component, hier is voor gekozen omdat er geen navigatie in de app zit en het scherm niet uit zoveel onderdelen bestaat dat dit opgesplitst dient te worden in meerdere componenten. In de homecomponent gebeurt niet gek veel, in de constructor worden er twee services aangeroepen waar de logica van de app in staat.

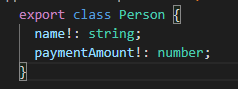


Figuur : Het aanroepen van de twee services

### Models

Er zijn twee models gebruikt voor het algoritme van de app, person en payment.

In de klasse person wordt bijgehouden welke personen er deelnemen aan de verreken app. Een persoon heeft een naam om weer te geven en een paymentAmount, hierin wordt bijgehouden hoeveel deze persoon heeft betaald.



Figuur : Het person model

In de klasse payment worden vier properties bijgehouden: name, paymentAmount, personThatPayed, payers.

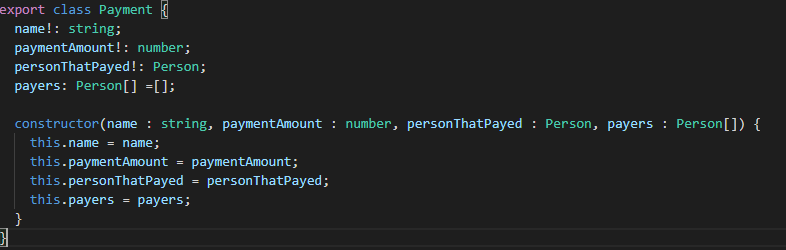
In de property name word bijgehouden waarvoor de betaling is, dit wordt in de app nu niet gebruikt, maar kan gebruikt worden om later uit te breiden.

In de property paymentAmount wordt bijgehouden hoeveel het totaal bedrag van de betaling is.

In de property personThatPayed wordt bijgehouden wie de betaling heeft voorgeschoten (de persoon die nog geld terugkrijgt).

In de property payers worden alle personen gezet die nog terug moeten betalen, er is gekozen om dit te doen zodat er later nog gekozen kan worden om mensen niet mee te laten betalen, dit is ook voor uitbreidbaarheid van de app.

Let op, zowel de property personThatPayed als payers zijn van het type Person, de klasse die waar we hiervoor over hebben gesproken.

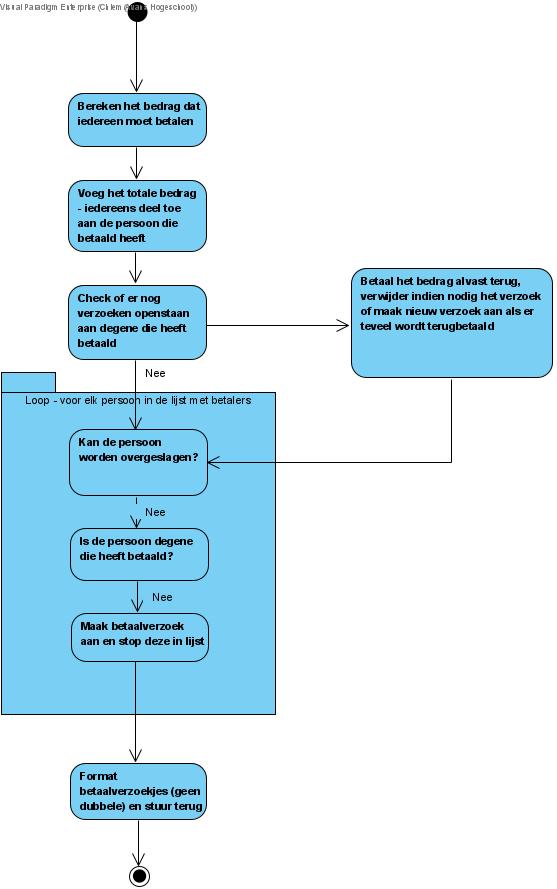


Figuur : Het payment model

### Algoritme

Het algoritme staat in de settle-payment service. Via het diagram hieronder wordt uitgelegd hoe het algoritme werkt, ik kan het ook in tekst uitleggen maar ik denk dat de afbeelding voor zich spreekt.

Let op: uit de afbeelding is niet duidelijk te zien dat dit algoritme wordt uitgevoerd voor ALLE betalingen in de lijst, niet voor een enkele betaling. Er staat dus eigenlijk een grote foreach loop om alle blokjes behalve de laatste, het formatten.



Figuur : Het algoritme uitgelegd in een diagram

# Bijlage

## Bijlage 1: Algortime in code

calculatePayment(payments : Payment[], persons : Person[]) : Payment[] {

    let paymentRequests : Payment[] = [];

    let excludeList : Person[] = [];

    payments.forEach(payment => {

      payment.payers = persons;

      let amountToPay = payment.paymentAmount / persons.length;

      payment.personThatPayed.paymentAmount += payment.paymentAmount - amountToPay;

      //Staan er nog verzoeken open

      let openPayment = paymentRequests.find(x => x.payers[0] == payment.personThatPayed); //

      while (openPayment != null) {

        if(openPayment != null) {

          console.log(openPayment);

          //het bedrag wordt ervan afgehaald

          let amount = openPayment.paymentAmount - amountToPay;

          console.log(amount);

          if(amount > 0) {

            paymentRequests.splice(paymentRequests.indexOf(openPayment), 1);

            let paymentRequest : Payment = new Payment(payment.name, amount, openPayment.personThatPayed, [openPayment.payers[0]]);

            paymentRequests.push(paymentRequest);

            excludeList.push(openPayment.personThatPayed);

            break;

          }

          else if(amount < 0) {

            paymentRequests.splice(paymentRequests.indexOf(openPayment), 1);

            let paymentRequest : Payment = new Payment(payment.name, -amount, payment.personThatPayed, [openPayment.personThatPayed]);

            paymentRequests.push(paymentRequest);

            excludeList.push(openPayment.personThatPayed);

          }

          else {

            paymentRequests.splice(paymentRequests.indexOf(openPayment), 1);

          }

        }

        openPayment = paymentRequests.find(x => x.payers[0] == payment.personThatPayed);

      }

      persons.forEach(person => {

        //Er word een betaalverzoek gemaakt voor iedereen behalve chiem en Jelmer (Exclude meerdere mensen! TODO!)

        let canSkipPerson : boolean = false;

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

          if(excludeList[i] == person) {

            canSkipPerson = true;

            break;

          }

        }

        if(!canSkipPerson) {

          if(person != payment.personThatPayed) {

            let paymentRequest : Payment = new Payment(payment.name, amountToPay, payment.personThatPayed, [person]);

            paymentRequests.push(paymentRequest);

          }

        }

      });

      excludeList = [];

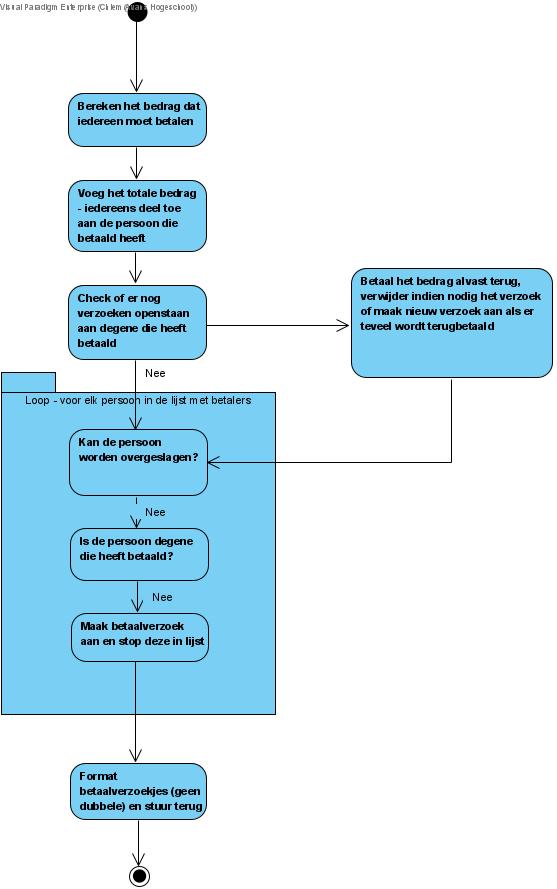
    });

    paymentRequests = this.formatPaymentRequests(paymentRequests, persons);

    return paymentRequests;

  }

## Bijlage 2: Algoritme als diagram



# Bibliografie

*guide/subject*. (2021, 03 28). Opgehaald van rxjs-dev.firebaseapp.com: https://rxjs-dev.firebaseapp.com/guide/subject

*observables in angular*. (2021, 03 28). Opgehaald van angular: https://angular.io/guide/observables-in-angular

*operators*. (2021, 03 28). Opgehaald van rxjs-dev.firebaseapp.com: https://rxjs-dev.firebaseapp.com/guide/operators