Technisch Ontwerp

Opdracht	Buddy App v3
Organisatie	Superbuddy
Opdrachtgever	Tim van den Heuvel
Schoolbegeleider	Tijn Witsenburg (WGM06)

Auteurs:

Nicander Mohrmann	s1103064
Silvan Otten	s1043664
Bart van Zanten	s1104266

Hogeschool Windesheim
ICT Software Engineering

Versie: 1.0
Datum: 21-06-2018

Samenvatting

Voor SuperBuddy wordt de huidige applicatie voor Buddies vernieuwd. Vanaf de grond af aan wordt de Buddy App v3 gemaakt met NativeScript en Angular. De applicatie draait hierdoor cross-platform op iOS en Android.

Er wordt gebruik gemaakt van de bestaande API om met de database te communiceren, waarbij nieuwe calls eventueel wel door ons gemaakt zullen worden.

De opdracht richt zich grotendeels op het realiseren van het orderpicking proces voor de bezorgers. Hierbij is de insteek dit zo soepel, intuïtief en efficiënt mogelijk te laten verlopen. Het liefst zouden Buddies de app kunnen downloaden en er meteen mee overweg kunnen.

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.

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 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 eind van ons project leveren we een minimal viable product op waarmee de gebruiker het gehele orderproces kan doorlopen. Dit houdt in: het inzien van open orders, orders accepteren, het inzien van order details, producten afvinken en de klant een rating kunnen geven.

Wat we niet zullen doen is: push notifications implementeren, geaccepteerde order slimmer maken en producten sorteren op winkel lay-out.

1.3 Afhankelijkheden

Vanuit het bedrijf is ons opgelegd om met Linux te werken. De bestaande dev environment werkt hier het best op. Voor het ontwikkelen van de app wordt NativeScript gebruikt in combinatie met Angular. Hierbij zijn we afhankelijk van de bestaande API, database en dockers. De API is gemaakt in Loopback en bevat een SDK builder die geïmplementeerd zal worden in de app.

1.4 Documentstructuur

1 Inleiding
Introductie die het project en dit document beschrijft.

2 Systeemoverzicht
Geeft een overzicht van het systeem in grote lijnen.

3 Systeemontwerp
Bevat de richtlijnen die aangehouden worden om het systeem te ontwerpen en te realiseren.

4 Componenten
Toont een overzicht van het gehele systeem en gaat in op individuele componenten.

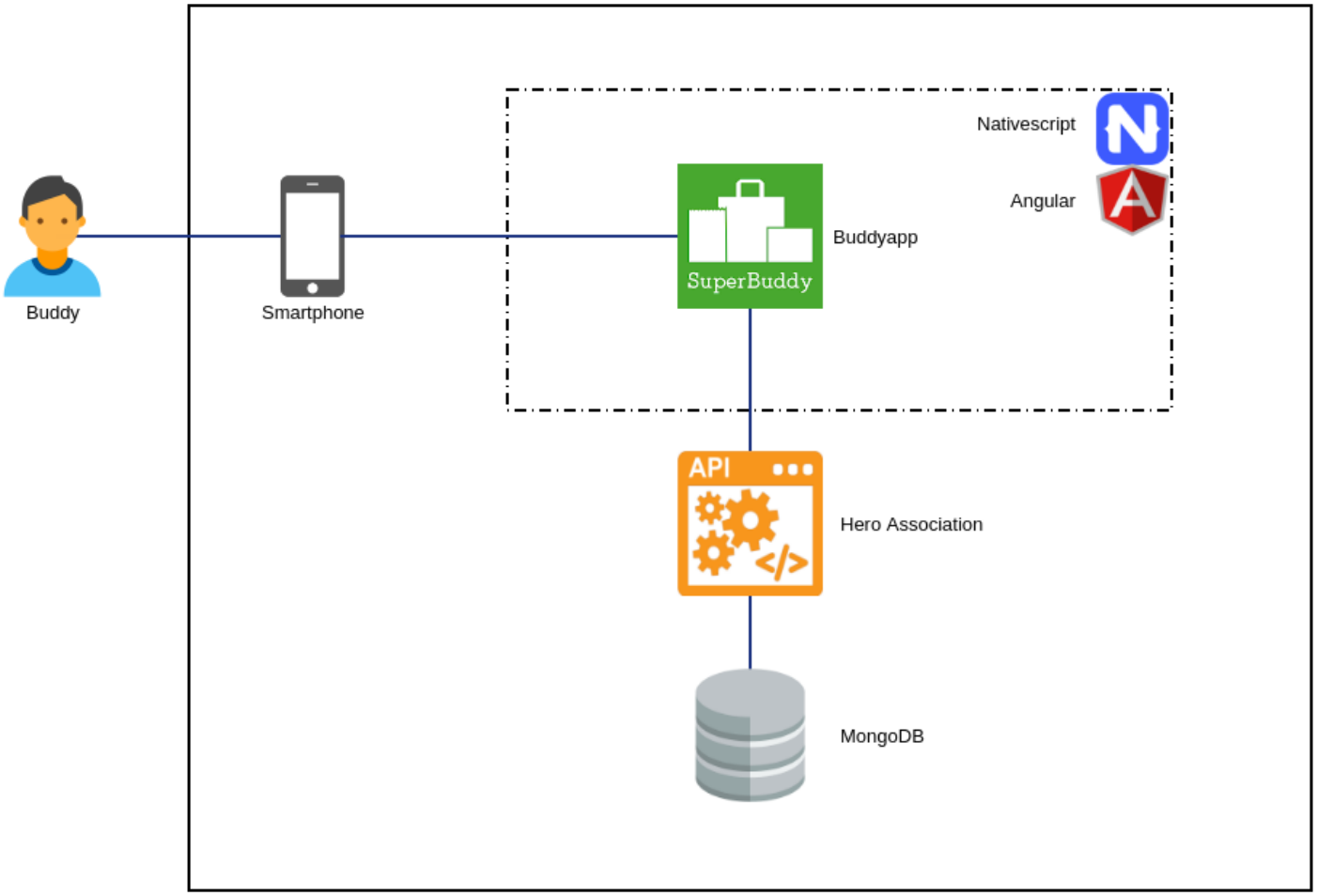
5 Bronnen
Geeft de bronnen die gebruikt zijn voor dit document.

2 Systeemoverzicht

Dit hoofdstuk geeft een beknopte introductie van de context van het systeem en de inrichting hiervan. Daarbij wordt de bestaande achtergrond van het project uitgelegd en volgen overwegingen die spelen bij de realisatie.

2.1 Context

2.1.1 Applicatie



Figuur 1

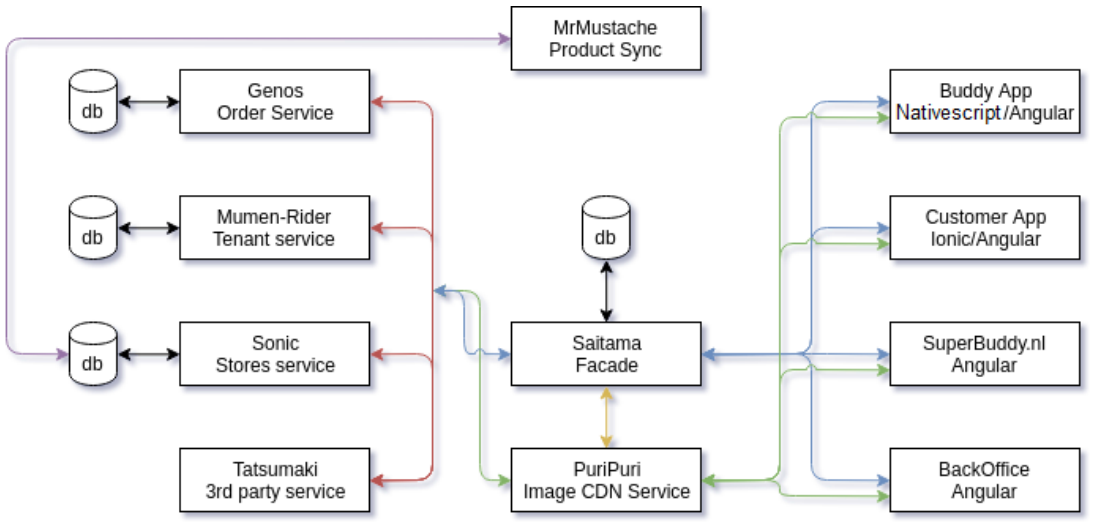
De app die wij gaan maken, BuddyApp v3, bestaat uit meerdere onderdelen. Alles wat binnen de stippellijn valt, valt binnen de scope van ons project. Dit is dus alleen de app zelf. Buiten de app om is er een database die we kunnen aanroepen via de API (Hero Association). Deze bestaat al en hoeft dus niet gemaakt te worden.

Het kan zo zijn dat benodigde calls naar de database nog niet bestaan in de Hero Association, waardoor deze toch door ons aangemaakt moeten worden.

Klanten die willen bestellen doen dit niet via de BuddyApp en dit valt dus buiten onze opdracht.

2.1.2 Omgeving

Onze applicatie zal gebruik maken van de Hero Association. Deze dient als tussenlaag voor communicatie met verschillende databases. Deze laag wordt ook door de overige producten/diensten binnen het bedrijf gebruikt.



Figuur 2

De kleuren van de pijltjes corresponderen met de verschillende applicaties en services. Saitama en PuriPuri kunnen elk aansluiten op alle services, en verbinden alles met elkaar.

De benodigde gegevens voor dit project zijn in verschillende databases ondergebracht. Deze hebben elk een eigen API die verbonden zijn met Saitama (blauwe lijnen). Om een request te maken zal je eerst authentication token moeten krijgen door in te loggen. Hieronder volgt een overzicht waar de services voor dienen.

Genos

Order, Cart & Payments service

1. Orders
    1. Status
    2. Times
    3. ProcessData
    4. Tenant link
2. Payments
    1. Tenant link
3. Cart

Mumen-Rider

Tenant service for user data

1. Orders
    1. Detail
    2. Products
2. Payments
    1. Detail
3. User Authentication
4. Budget logs

Sonic

Store & Buddy service

1. Buddy
    1. Availability
    2. Detail
    3. Order actions
2. Cities
    1. Areas
3. Stores
    1. Areas
4. Product
    1. Elastic Search

Tatsumaki

3rd party communications service

1. Slack
2. Yuki
3. Mixpanel
4. Salesforce
5. SMS
6. Email

PuriPuri

Image CDN Service

1. Image saving
2. Image manipulation
3. Image viewing

Saitama

Facade and customer service

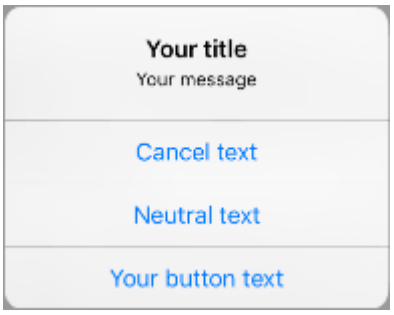
1. Open to world
2. Keeps track of customer & User relations
3. Role & Auth management
4. Connects all services to frontend

Figuur 3

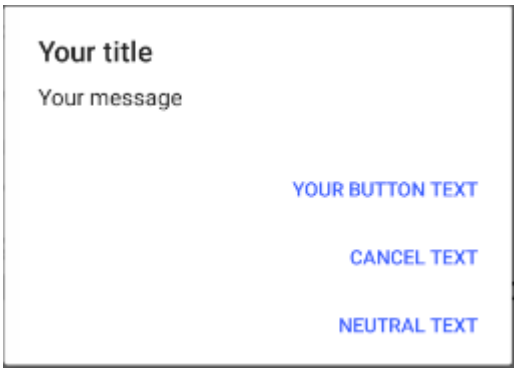
2.2 Karakteristieken

De App is gemaakt voor Android en iOS waarbij er met NativeScript gewerkt wordt. Er bestaat dan een code basis die door beide operating systems gebruikt kan worden. De code wordt dus naar native componenten omgezet. Dit betekent dat de app er op beide platforms soms anders uit komt te zien, of dat bepaalde functionaliteiten kunnen verschillen. Het grootste voordeel hiervan is dat je maar een codebase hoeft te onderhouden.

iOS Confirm Dialog



Android Confirm Dialog



Figuur 4

De Buddies maken dagelijks gebruik van de applicatie op hun telefoon. Zij zijn niet altijd technisch ingestelde mensen, dus de applicatie zal met zo min mogelijk uitleg intuïtief moeten werken. Buddies zijn al gewend aan de huidige App en veranderingen moeten aansluiten bij hun verwachtingen en/of wensen van het bedrijf. Hierbij maken zij gebruik van hun eigen internetverbinding die nodig is om de app te gebruiken.

Daarnaast zijn het ook gewoon mensen die fouten kunnen maken, welke zoveel mogelijk voorkomen of hersteld moeten worden. Er wordt vanuit gegaan dat zij niet welwillend kwaadaardig handelen. De foutafhandeling zal dus gemakkelijk te begrijpen zijn.

De buddies hebben in principe toegang tot alle eigen gegevens, orders die af te handelen zijn of die ze zelf afhandelen. Er wordt nog bekeken welke klantgegevens wel of niet in te zien zijn.

Het bedrijf zet erg in op growth, waardoor met een groeiend aantal gebruikers rekening gehouden moet worden. Dit houdt onder andere in dat calls naar de database efficiënt ingericht dienen te zijn en dat er gedacht wordt aan de data die heen en weer gestuurd wordt.

2.3 Architectuur

Er is één actor die de BuddyApp gebruikt, buddies, ondanks dit kan een superadmin ook overal bij, indien nodig.
Een buddy maakt eerst een account aan op de website van superbuddy, deze moet. Vervolgens moet de BuddyApp gedownload worden middels de play store op android of de appstore op ios. Als de buddy geverifieerd is kan er ingelogd worden in de app. Hiervoor is een internetconnectie nodig, die stabiel moet blijven tijdens het hele orderprocess.
De app zal gehost worden op amazon servers.

Wij werken met een component based architecture, wat betekent dat als er een nieuwe pagina moet komen, deze in een component komt. Een component bestaat uit meerdere onderdelen die worden uitgelegd in hoofdstuk 3. De data access layer gaat via een SDK die automatisch gegenereerd wordt door middel van een code first JSON database.

2.4 Infrastructuur

2.4.1 Security

Omdat de app publiekelijk zal worden gemaakt en geplaatst wordt in de appstore moet er wat gedaan worden aan security. Allereerst zal er een inlog systeem toegevoegd worden aan de app zodat niet zomaar iedereen het systeem binnen kan komen. Na een succesvolle inlogpoging zal er een authenticatie sleutel aangemaakt worden die ervoor zorgt dat de ingelogde gebruiker toegang heeft tot bepaalde API calls. De authenticatie per API call wordt geregeld binnen de API en valt buiten de scope van het project.

Naast het implementeren van een login systeem zal de app beveiligd worden door sign encryption. Dit is verplicht voor het publiceren van een app in de stores.

2.4.2 Error Handling

Omdat er veel API calls gemaakt zullen worden in de app willen we op een centrale plek alle errors afvangen, zo kunnen we door het gehele project consistente error berichten weergeven. Naast dat de error berichten consistent zijn hoeft er niet voor elke API call een error toegevoegd worden mits het fout gaat. Buiten dat om zorgt dit er ook voor dat alle errors afgevangen worden, zo voorkom je dat er een mogelijkheid is dat de app een error geeft zonder dat je er iets van merkt. Het toepassen van globale error afhandeling was een aanvraag vanuit het dev team aangezien zij een gelijksoortige aanpak hebben in de andere applicaties.

2.5 Testen

Om te testen worden testscenario's opgezet die de acceptatiecriteria goed encapsuleren, gedocumenteerd in het functioneel ontwerp. De tests voert degene die verantwoordelijk is voor de userstory uit.

Het development team zelf hoort de daadwerkelijke testplannen niet uit te voeren noch op te stellen, deze worden ingericht door overige medewerkers binnen het bedrijf. Aangezien deze buiten onze scope plaatsvinden komen de resultaten niet binnen onze documentatie.

3. Systeemontwerp

Dit hoofdstuk dient om voldoende informatie aan developers te geven om het systeem correct in te richten. Om het te produceren kan gekeken worden naar hoofdstuk 4.

3.1 Ontwerpmethoden en standaarden

Voordat er begonnen wordt met programmeren moet er eerst een schermontwerp worden gemaakt, deze wordt naar de product owner gestuurd met de vraag voor feedback. Wel kan er worden gewerkt aan het functionele gedeelte van de code. Nadat het schermontwerp is goedgekeurd kan er ook aan de GUI worden gewerkt. Als er later blijkt dat er iets veranderd moet worden, dan moet dit ook goedgekeurd worden door de product owner.

Als er een nieuwe feature gerealiseerd moet worden komen daar een paar onderdelen bij te kijken. Als de nieuwe feature alleen iets nieuws is op een bestaande pagina, hoeven alleen de gebruikte imports worden toegevoegd aan de gebruikte module.
Als er een nieuwe pagina toegevoegd moet worden dan moet er een component, html pagina en scss file, ook moet de route naar de pagina worden toegevoegd.. Als de nieuwe component niet een bestaande module gebruikt, zal deze ook aangemaakt moeten worden. Als er functies of variabelen zijn die in meerdere components gebruikt worden moeten deze in een service geplaatst worden, die dan in module geïmporteerd kunnen worden.

3.2 Documentatie standaarden

Aangezien er binnen het bedrijf geen personen specifiek voor code verantwoordelijk zijn worden geen authors aangegeven. Documentatie, net als overige code, zal in het Engels geschreven zijn.

Er wordt zoveel mogelijk gemaakt van standaard C++ comments (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

// NativeScript

// Services

// Dependencies

// Components
```

```
// SDK
```

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

```
// Sets status to checked or unchecked depending on where it's swiped from.
    public async onSwipeLeft(args: ListViewEventData, checked: boolean) {
 ...
    }
```

3.3 Programmeerstandaarden

Er zijn een aantal standaarden waar wij ons aan moeten houden vanuit het bedrijf en Angular. Deze zijn ondergebracht in projectstructuur - hoe ons project is opgebouwd, en in enkele codestandaarden.

3.3.1 Projectstructuur

Voor mappenstructuur en files houden wij de structuur van Angular aan. Dit houd in dat pagina's onder de folder "pages" komt. Elke pagina (component) bevat een HTML, Sass, ts, module en routes file. De HTML en Sass files worden gebruikt voor de visualisatie van de pagina. TS file is voor de logica binnen de pagina. Module file is voor het importeren van gebruikte componenten (denk hierbij aan plugins). Routes file wordt gebruikt voor het aanmaken van een route binnen het project. Voor custom services en components zullen er aparte folders zijn buiten de "pages" folder.

Voorbeeld:

- components
    - rate
        - rate.component.html
        - rate.component.ts
- pages
    - login
        - login.component.html
        - login.component.scss
        - login.component.ts
        - login.module.ts
        - login.service.ts
- services
    - api.service.ts
    - error.service.ts

In het hierboven gegeven voorbeeld is te zien dat er ook standaarden zijn voor de naamgeving van bestanden. Bestandsnamen houden de standaarden aan van Angular. Dit houd in dat een bestandsnaam altijd begint met de inhoud of functie van het bestand. Als de naam bestaat uit twee of meerdere woorden zal er camelCase toegepast worden Bijvoorbeeld: "login". Dit wordt gevolgd door de type van het bestand zo heb je, components, directives, services, routing en modules. Bijvoorbeeld: "login.module".

3.3.2 Code standaarden

Omdat er gebruik wordt gemaakt van JS en TS zijn er vrije keuzes voor standaarden. Het bedrijf houdt het volgende aan:

- 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
- “Let” wordt gebruikt voor variabelen die niet direct een waarde toegewezen krijgen
- SDK imports krijgen de variabelenaam “naamSDK”
- Service imports krijgen de variabelenaam “naam”
- Voor API calls wordt er gebuikt gemaakt van async en await
- Variabelen krijgen altijd een type mee. “let message: string = …”
- Als er gebruik gemaakt wordt van variabelen in de HTML wordt de volgende annotatie gebruikt: “<Label [text]=’message’></Label>” en niet “text={{message}}”

3.4 Software Development Tools

Om het ontwikkelen van onze app hebben we de volgende tools nodig:

Docker

- De gebruikte API draait al binnen een docker en er zal een docker gemaakt worden voor de app

Visual Studio Code

- Dit is de go-to editor binnen het bedrijf. Visual Studio Code bevat goede support voor TypeScript en heeft een groot aanbod van extensies

Studio 3T

- Studio 3T is een tool om de nosql database in te kunnen zien. Hiermee kunnen wij een connectie maken met onze local database.

GitKraken

- De go-to git client binnen het bedrijf.

Jira

- Issue tracker. Hier staat onze backlog en sprint backlog.

Bitbucket

- Hier staat al de code. Binnen bitbucket zal er gebruik gemaakt worden van pull request. Hiermee kunnen wij aangeven dat wij onze aanpassingen in de main branch willen toepassen. Deze pull requests worden nagekeken door de lead developer.

Android Studio

- Voornamelijk gebruikt om gemakkelijk een emulator op te zetten en de performance van de app te monitoren.

3.5 Openstaande issues

Na de oplevering bij het bedrijf spelen er alsnog een aantal zaken die we niet naar wens konden oplossen, waar we niet aan toegekomen zijn en enkele punten die buiten de scope vallen. Deze worden hier verder toegelicht.

3.5.1 Verbeteringen

Lazy Loading

De out of the box compile methode kan nog niet goed overweg met global modules hierdoor werkt lazy loading niet optimaal. Een mogelijke fix is om een andere compile methode toe te passen zoals Webpack.

RadListView indexering

In de RadListViews waar headers in toegevoegd worden is de indexering van de items niet optimaal. De headers worden namelijk meegenomen in deze indexering. Hierdoor corresponderen de indexen van de items niet meer met de RadListView.

Om dit op te lossen wordt nu voor de items een nieuwe lijst aangemaakt waar ook de headers in zitten. Deze komt dan wel weer overeen met de RadListView, maar moet ook telkens bijgehouden worden indien de items veranderen.

Er moet dus nog gekeken worden of hier een betere uitwerking voor gevonden kan worden.

MenuBar swipe & ActionBar buttons

Op dit moment vallen de buttons in de ActionBar (deels) over de swipe boundaries voor de MenuBar. Hierdoor wordt het gewenste gedrag verstoord. Mooi zou zijn om de swipe boundaries niet op de ActionBar te laten gelden, maar hier was geen mogelijkheid voor gevonden.

3.5.2 Open punten

Er zijn nog een aantal zaken waar we niet aan toegekomen zijn, die wel voor het live gaan van de applicatie gewenst zijn.

Profiel pagina

Hier kunnen buddies hun eigen profiel inzien en indien gewenst aanpassen.

Live Refresh open orders

Indien er nieuwe open orders binnenkomen die een Buddy kan accepteren hoort hij hiervan op de hoogte gesteld te worden d.m.v. een push notification. Dit kan dus voor meerdere Buddies gelden.

Live Refresh accepted orders

Een klant kan nu bij een geaccepteerde order nog eenmalig maximaal vijf producten toevoegen. Hier dient voor de Buddy die owner is een push notification te komen en als hij de order open heeft hoort een dialog te openen.

Password reset

Indien een Buddy zijn wachtwoord vergeten is moet hij deze nog kunnen resetten.

Order delivery view

Een order kan nu nog niet in delivery komen.

Order history

De Buddy krijgt hier een lijst met completed orders te zien.

3.5.3 Toekomstige punten

Het belangrijkste lange termijn doel is het verbeteren van het orderproces zodat Buddies efficiënter kunnen werken. Onderstaande punten zijn besproken, maar er dient vooral gekeken te worden naar hoe er tijdwinst geboekt kan worden voor de Buddies.

Gamification

Het is een wens van de product owner om gamification toe te passen ter stimulatie van de Buddies. De invulling hiervan staat nog open.

Slimme orderpicking

Het versnellen van het orderproces kan verder geholpen worden door een slimmere orderpicking. Dit zou terug kunnen komen in een slimme lijst met te halen producten op volgorde van winkelindeling, of kan er gekeken worden naar welke producten lang duren om op te halen.

Meerdere orders combineren

Het liefst zou je als Buddy meerdere orders tegelijkertijd kunnen afhandelen zodat je niet op en af naar de supermarkt(en) hoeft te gaan.

Slimme routebepaling

Indien er meerdere orders zijn is het ook belangrijk om een goede route naar de klanten hierbij te maken voor de Buddy zodat hij deze gemakkelijk kan volgen.

4 Componenten

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

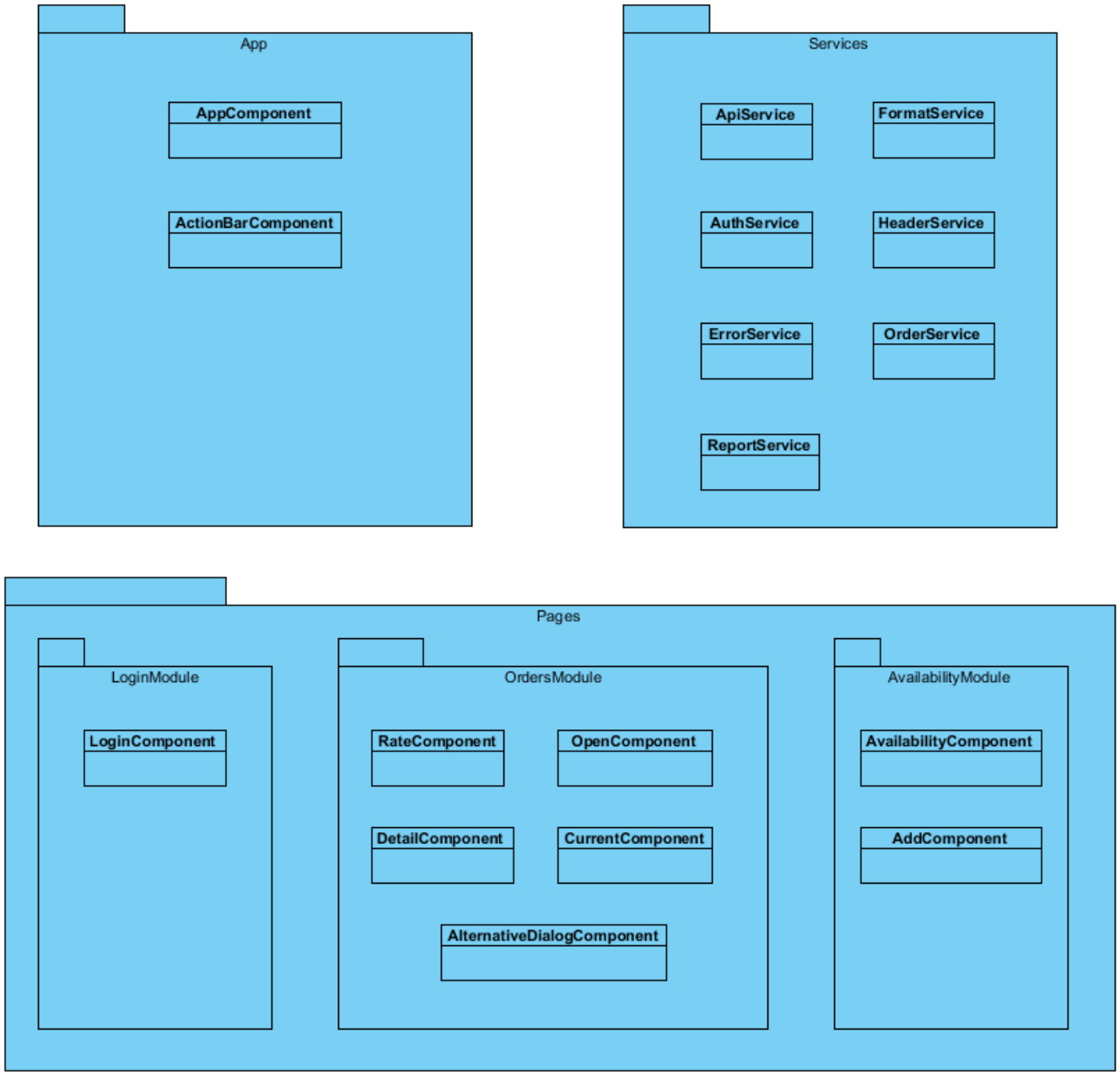
De AppComponent en ActionBarComponent zijn essentieel door de hele applicatie heen. Deze zorgen ervoor dat er correct genavigeerd kan worden met de juiste instellingen.

Services zijn aan te roepen om ervoor te zorgen dat er op een uniforme manier om wordt gegaan met bijvoorbeeld het formatten van gegevens, of het opslaan van autorisatie gegevens.

4.1 Hoofdlijn

De applicatie dient overzichtelijk gestructureerd te zijn. Dit komt terug in de locatie van de klassen. Deze zijn ondergebracht in verschillende folders. Elk is voor zijn eigen gedeelte verantwoordelijk, maar kan gebruik maken van anderen. Dit is terug te zien in figuur 5.

Wanneer herhaaldelijk dezelfde functionaliteit gebruikt wordt dient dit ondergebracht te worden onder een service. Hierdoor blijft het programma uniform en ontstaat er geen redundante code. De pagina's zelf zijn gegroepeerd op modules waar zij onder vallen. Hoe de componenten met elkaar samenwerken blijkt uit de individuele componenten.



Figuur 5

4.2 Individuele componenten

Onder dit kopje volgen de individuele componenten die de bouwstenen van de applicatie vormen. Hiermee hoort duidelijk te worden hoe alles in elkaar hoort te steken.

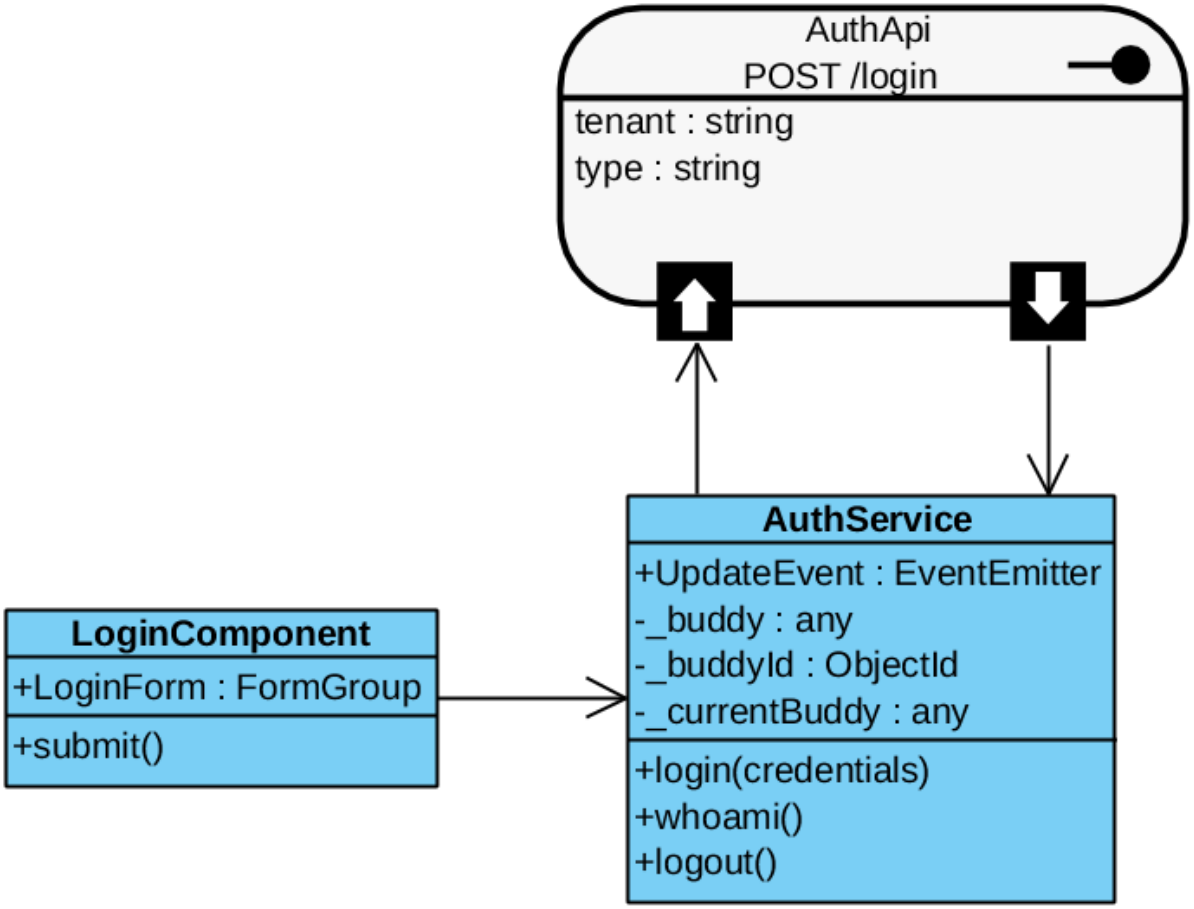
4.2.1 Login Component

User story BUA-27: Login view

Doel

Dit component wordt gebruikt als landingspagina in de app. Op deze pagina kunnen gebruikers inloggen met hun gegevens zodat zij daarna de gewenste order informatie kunnen bekijken.

Klassendiagram



Functies

- **submit**
  Door middel van deze functie wordt er een API call gemaakt met de ingevulde inloggegevens van de gebruiker. Na een succesvolle poging zal de gebruiker doorgestuurd worden naar een andere pagina

- **isLoggedIn**
  Deze functie kijkt of de gebruiker al eerder ingelogd is geweest, zo ja dan wordt de gebruiker gegevens opgehaald en doorgestuurd naar een andere pagina
- **logout**
  Functie waarmee de gebruiker uitgelogd wordt

Dependencies

- Imports
  - Dialogs uit “ui/dialogs”
  - RouterExtensions uit “nativescript-angular/router”
  - LoadingIndicator uit “nativescript-loading-indicator”

Bewerkingen

- Inlog pogingen worden bijgehouden in de database

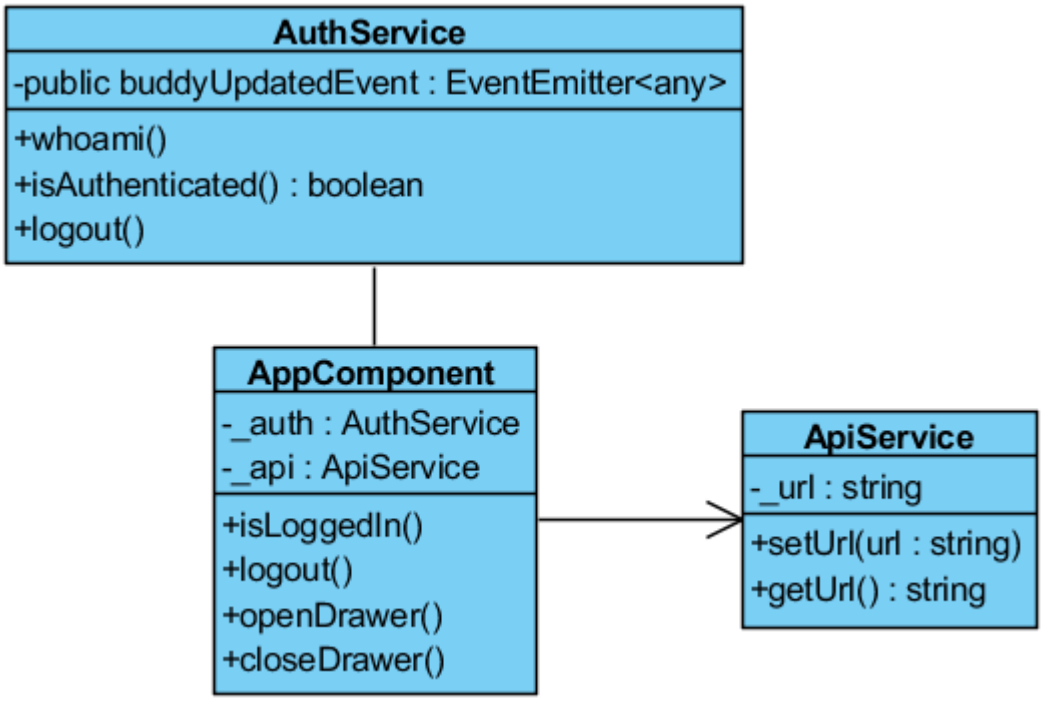
4.2.2 App Component

User story: BUA-30 Sidebar menu

Doel

App component is de entry file van het project, hierin worden belangrijke instanties gestart. Daarnaast bevat de app component de side menu

Klassendiagram



Functies

- **isLoggedIn**
  Deze functie checked of de gebruiker al een keer eerder heeft ingelogd, zo ja wordt de gebruiker automatisch ingelogd en doorgestuurd naar open orders
- **logout**
  Uitlog functie voor de side menu
- **openDrawer**
  Opent de side menu
- **closeDrawer**
  Sluit de side menu

Dependencies

- Imports
  - NativeScriptUISideDrawerModule uit "nativescript-ui-sidedrawer/angular"
  - RadSideDrawerComponent uit "nativescript-ui-sidedrawer/angular"
  - RadSideDrawer uit "nativescript-ui-sidedrawer"
  - RouterExtensions uit "nativescript-angular/router"

4.2.3 Actionbar Component

User story: BUA-41: Custom header

Doel

Met deze component kunnen we een custom actionbar toepassen op de pagina's.

Functies

- **showDrawer**
  Opent de side menu door middel van een tap (klik)
- **navPop**
  Navigeert naar de pagina waar je zojuist vandaan komt

Dependencies

- Imports
  - RadSideDrawerComponent uit "nativescript-ui-sidedrawer/angular/side-drawer-directives"
  - RouterExtensions uit "nativescript-angular/router"

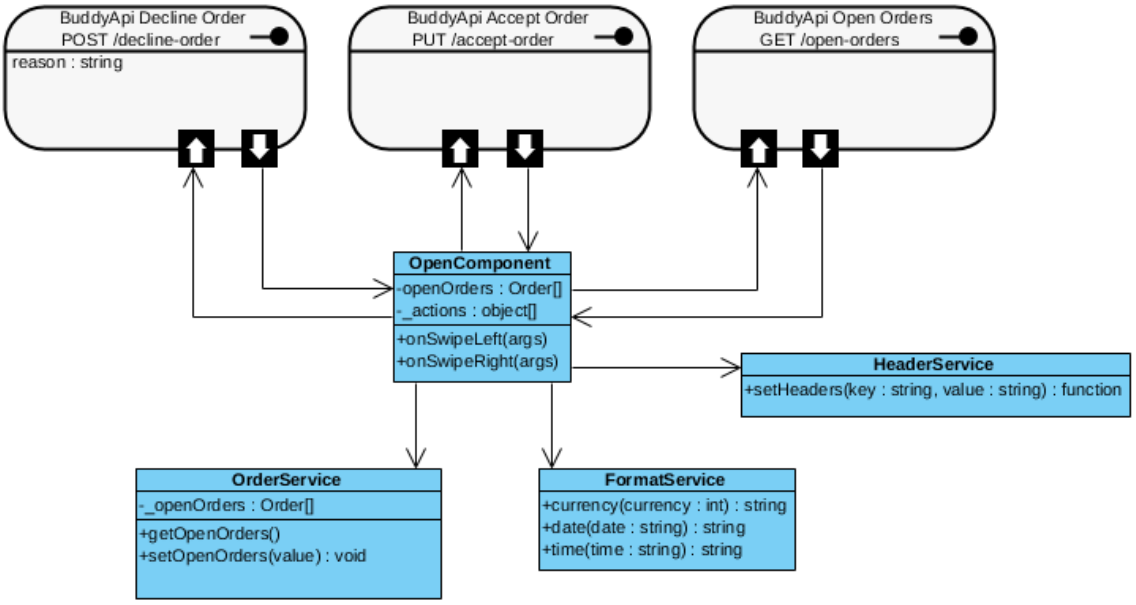
4.2.4 Open Component

User story BUA-29: Open order view
User story BUA-38: Change order status

Doel

Dit component wordt gebruikt om de open order data uit de database te halen en deze weer te geven in de GUI. Ook moeten orders geswiped kunnen worden om te accepteren/afwijzen.

Klassendiagram



Er worden vanuit de services OrderService, FormatService en HeaderService functies gebruikt in OpenComponent. Tevens worden de opgehaalde orders opgeslagen in de OrderService. De orders worden opgehaald en geaccepteerd/afgewezen door API calls te maken naar de backend.

Dependencies

De imports die nodig zijn voor dit component zijn:
Dialogs uit “ui/dialogs”
RadListView en alle aanhangende eventlisteners uit “nativescript-ui-listview”
RouterExtension uit “nativescript-angular/router”

Bewerkingen

Bij afwijzen wordt de rede en buddyId opgeslagen in een order
Bij accepteren wordt buddyId opgeslagen in een order en status naar inProgress gezet

Data

De json data die nodig is voor dit component is:
‘order._processData.subTotal'
'order.earliestDeliverTime'

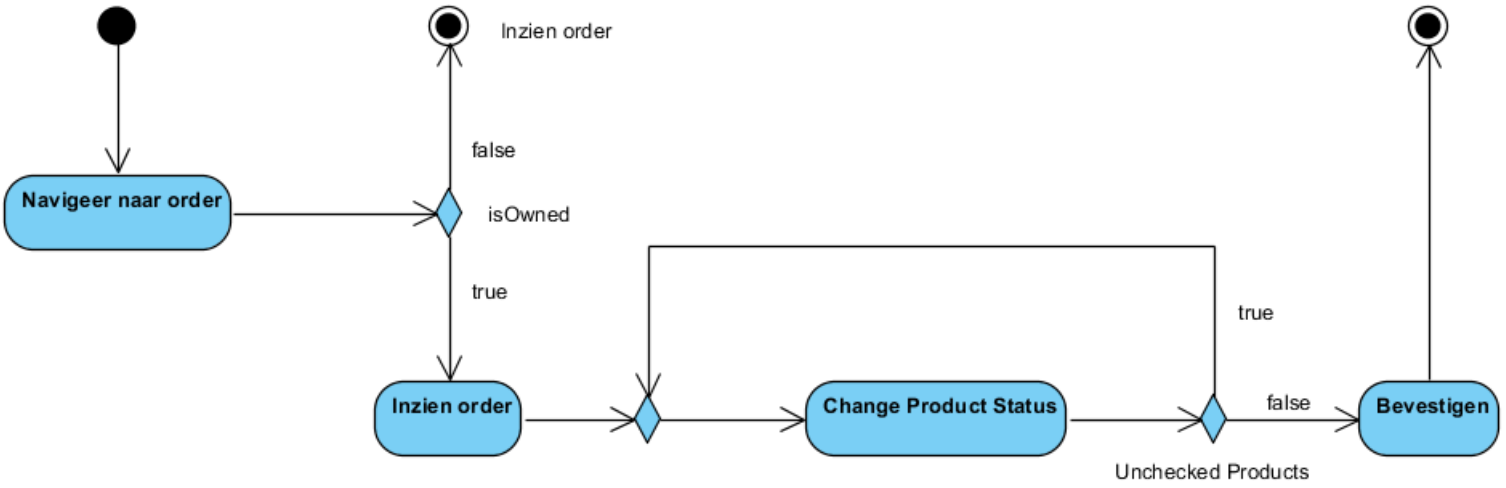
'order.latestDeliverTime'
'order._processData.productCount'
'order._processData.stores'
'order._properties.allowEarly'
'order.deliverDate'
'order._id'

4.2.5 Detail Componen

User story BUA-34: Order detail
User story BUA-35: Change product status

Doel

Dit component dient details van een order met status paid te tonen aan een Buddy. Hiermee kan hij bepalen of de order geschikt is om te accepteren. Indien de Buddy een order geaccepteerd heeft kan hij een order doorlopen. Elk product dient dan nagekeken te worden.

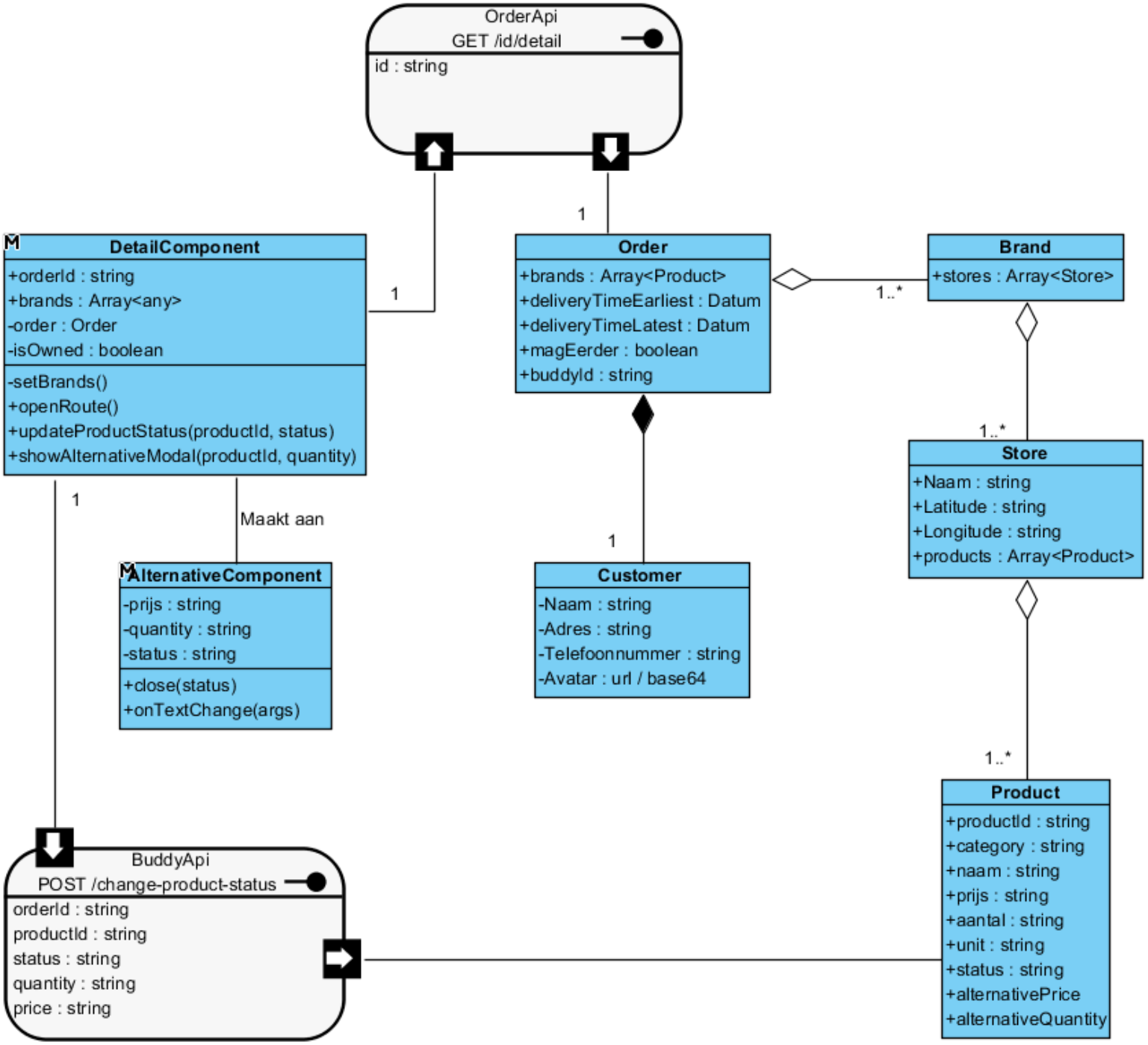


SuperBuddy

Klassendiagram

Een instantie van DetailComponent wordt aangemaakt wanneer er naar ~/orders/detail/:id wordt gegaan. Er moet om naar deze pagina te navigeren altijd een orderId meegegeven worden.

Deze haalt via de OrderApi een Order binnen, welke bestaat uit een klant (Customer) en brands. Binnen een brand vallen stores met bestelde producten.



De setBrands functie zorgt ervoor dat alle relevante informatie uit een order goed wordt geset in de brands variabele. Hierin worden producten verzekerd van een category en de brand krijgt de bijpassende naam.

De openRoute functie kan aangeroepen worden om met de adresgegevens van de klant een GoogleMaps url te construeren om deze vervolgens te openen.

De updateProductStatus zorgt ervoor dat de producten in de brands van de DetailComponent geupdate worden indien die veranderd is in door de callback van de showAlternativeModal. Hierdoor hoeft er geen nieuwe call gemaakt te worden naar de API.

De showAlternativeModalComponent roept een modal component aan welke de status van een product kan aanpassen. Indien deze naar alternatief wordt veranderd dient er ook een prijs en aantal in de callback te komen.

Dependencies
Voor het correct sorteren van brands en bijbehorende stores met producten wordt lodash gebruikt. Voor het notificeren van de Buddy wanneer een productstatus verandert is wordt Snackbar gebruikt.

Resources
Voor het tonen van de route wordt gebruik gemaakt van Google Maps. Hiervoor is de Developer Guide (Google 2018) aangehouden. Met de locaties van winkels en afleveradres wordt hiermee een cross-platform URL gebouwd die automatisch de route geeft in Google Maps.

Bewerkingen
De status, alternativeQuantity en alternativePrice van een product kunnen aangepast worden.

Data
Via de API wordt de gehele order detail binnengehaald. Deze omvat veel informatie die niet nodig is waardoor er gespecificeerd moet worden.

Als het nodig is de klant te raadplegen kan de Buddy, indien zijn buddyId overeenkomt met de buddyId van de order, gegevens van de klant inzien. Hij heeft dan toegang tot de naam, adres en avatar van de klant.

Om op het juiste moment bij de klant te komen heb je de volgende gegevens nodig
- Customer address
- deliveryTimeEarliest
- deliveryTimeLatest

Om duidelijk te krijgen welke producten in welke winkels gehaald kunnen worden
- stores met hun latitude en longitude
- products met hun gegevens

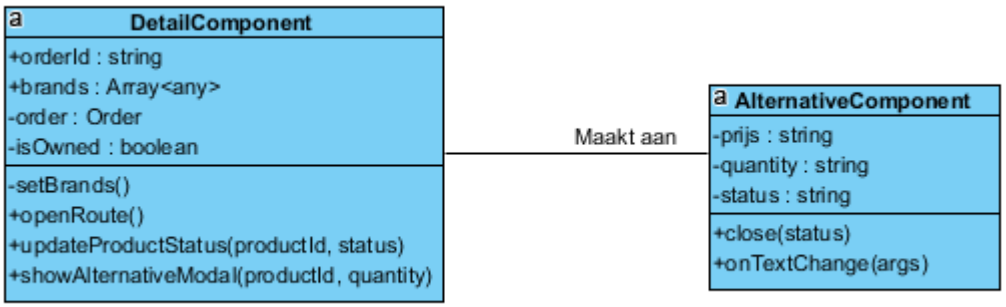
4.2.6 Alternative Component

User story BUA-35: Change product status

Doel

Een alternative component wordt aangeroepen wanneer de gebruiker de status van een product wilt veranderen naar alternative of notFound. Hierbij wordt het aantal van het gekozen product meegegeven voor het gemak van de Buddy.

Klassendiagram



De close(status) functie is de callback functie van de showAlternativeModal(productId, quantity). Deze heeft drie verschillende callback opties. Wanneer een modal dialog wordt gesloten krijgt de callback default undefined als resultaat terug. Dit wordt tevens bij de annuleer optie meegegeven.
Het product krijgt de status mee van notFound als het product niet gevonden is (zonder alternatief).
Wanneer alternatieve producten worden gekozen dient er een alternatieve prijs en aantal producten teruggestuurd te worden.

De onTextChange(args) krijgt het invoerveld van de alternatieve prijs binnen bij iedere wijziging en past deze aan zodat de Buddy gemakkelijk ermee overweg kan. Zo kan er geen negatieve waarde ingevuld worden, noch vreemde karakters.

Dependencies

Er wordt gebruik gemaakt van bepaalde validators in de FormGroup om te zorgen dat alleen valide data in de callback gegeven kan worden.
De onTextChange(args) maakt gebruik van een de native TextField en de setSelection hierop moet weten of het huidige platform Android of iOS is.

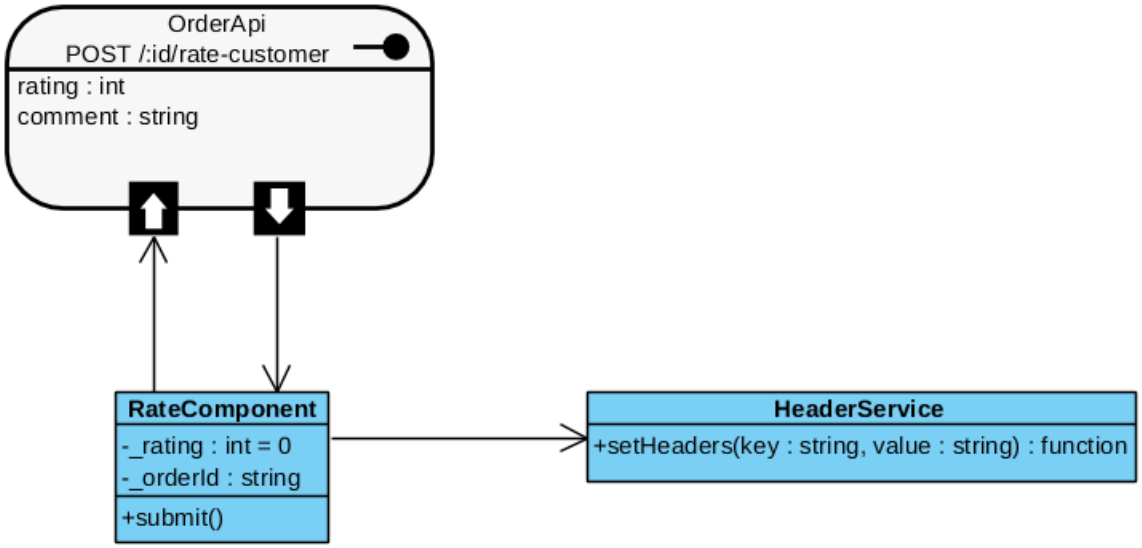
4.2.7 Rate Component

User story BUA-40: Rate customer

Doel

Dit component wordt gebruikt door de buddy om een beoordeling te geven aan een klant en vervolgens een order af te ronden.

Klassendiagram



Om de beoordeling op te slaan wordt er gebruik gemaakt van de rateCustomer functie in de SDK.

Dependencies

Om het orderId uit de route parameters te halen is de import ‘ActivatedRoute uit “@angular/router” nodig.

Bewerkingen

De rating en comment worden opgeslagen in order.properties.customerRating als object.

Data

De json data die nodig is voor dit component is:
‘order._id’

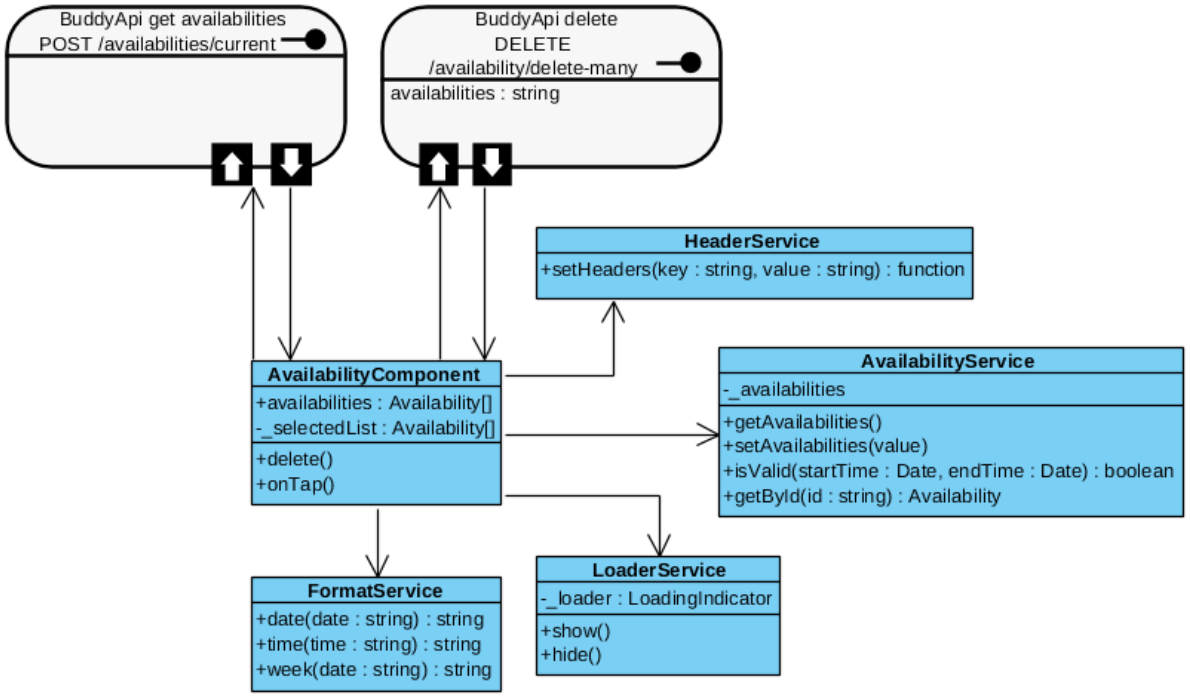
4.2.8 Availability Component

User story BUA-32: Availability view

Doel

Dit component zorgt er voor dat alle beschikbaarheden worden weergegeven in de GUI en er mogelijk is op deze aan te passen, verwijderen of toevoegen.

Klassendiagram



De onTap functie in AvailabilityComponent redirect naar het aanpassen van een beschikbaarheid.

Dependencies

De import ‘RouterExtension uit “nativescript-angular/router”’ is nodig om te kunnen linken naar andere pagina’s

Bewerkingen

Er worden beschikbaarheden uit de database verwijderd.

Data

De JSON data die nodig is voor dit component is:
‘availability.id’
‘availability.start’
‘availability.end’

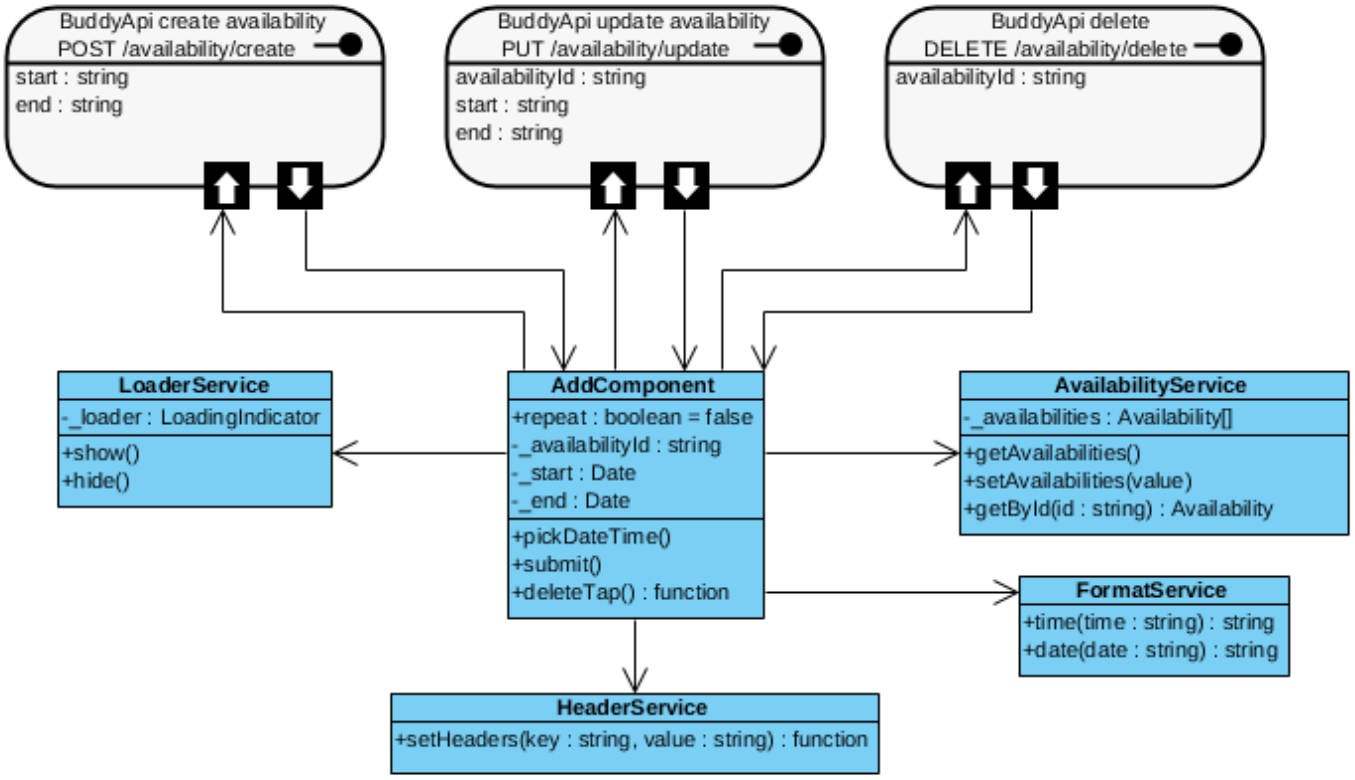
4.2.9 Add Availability Component

Dit component omvat de userstory Buddy Availability.

Doel

Dit component geeft de buddy een GUI om gemakkelijk een beschikbaarheid toe te voegen of aan te passen.

Klassendiagram



de deleteTap functie returned een functie voor de actionbar component om uit te voeren.

Imports

- RadSideDrawerComponent uit “nativescript-ui-sidedrawer/angular”
- RouterExtensions uit "nativescript-angular/router"
- ActivatedRoute uit “@angular/router”

Bewerkingen

Bij toevoegen van een beschikbaarheid wordt er een nieuwe availability aan de database toegevoegd met een starttijd, eindtijd en buddyId.
Bij het aanpassen worden de starttijd en eindtijd aangepast.
Bij het verwijderen wordt de beschikbaarheid op verwijderd gezet.

Data

De JSON data die nodig is voor dit component is:
‘availability.id’
‘availability.start’
‘availability.end’

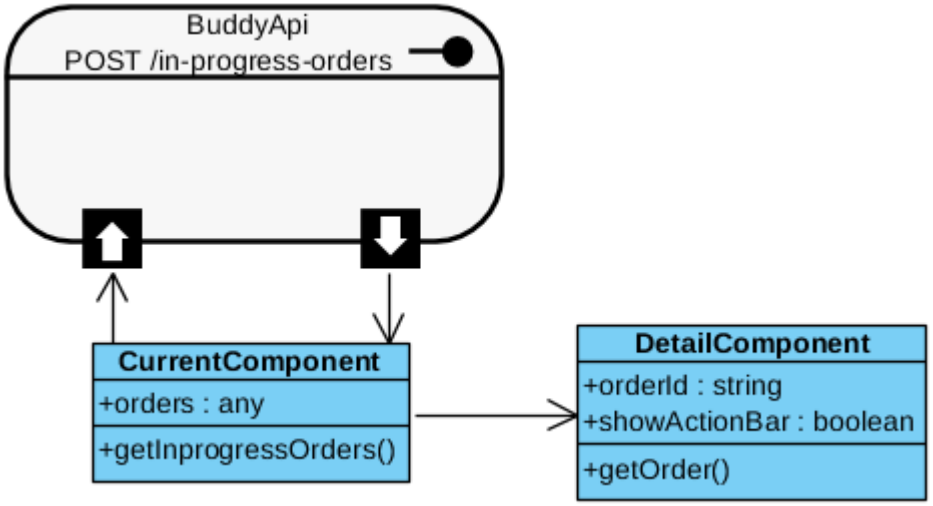
4.2.10 Current Component

User story BUA-38: Accepted orders view

Doel

Deze component moet het mogelijk maken om de geaccepteerde orders gemakkelijk in te zien. De gebruiker moet weinig handelingen verrichten om tussen de geaccepteerde orders te wisselen.

Klassendiagram



Functies

- **getInProgerssOrders**
  Haal de geaccepteerde orders op aan de hand van de unieke ID van de gebruiker

Data

De volgende data vanuit de API is nodig:
“order.displayId”
“order.id”

5 Bronnen

TJ VanToll (15 februari 2015)
How NativeScript works
Geraadpleegd op 22 mei 2018, van
https://developer.telerik.com/featured/nativescript-works/

Google (laatst bijgewerkt 14 juni 2018)
Developer Guide | Maps URLs | Developers
Geraadpleegd op 20 juni 2018, van
https://developers.google.com/maps/documentation/urls/guide

Google (2018)
Google C++ Style Guide
Geraadpleegd op 19 juni 2018, van
https://google.github.io/styleguide/cppguide.html#Comments