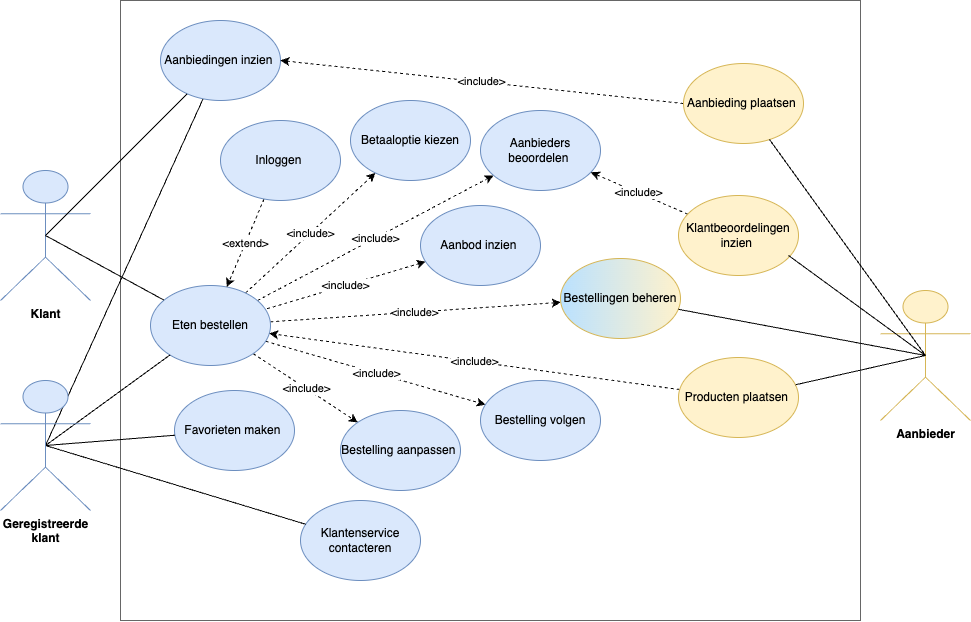
# **Deelvraag: Wat zijn de verschillende onderdelen van de mobiele applicatie**?

Voor het bepalen van de verschillende onderdelen die horden bij de te ontwikkelen mobiele applicatie moet eerst aandacht worden geschonken aan de functionaliteit die wordt verwacht van de applicatie. Hierbij wordt eerst middels usercases en userstories de gewenste functionaliteit geschetst. Hierna worden de verschillende functionele en niet-functionele aspecten beoordeeld. Tot slot wordt het C4 model voor de Homecooked applicatie geschetst. Aan de hand hiervan wordt er een goed beeld geschetst van de benodigde onderdelen van de applicatie.

# **Usecase Diagram**

Onderstaand is het usecase digram te zien van de homecooked mobiele applicatie. Hierbij wordt een onderscheid gemaakt tussen de klanten die eten bestellen en de aanbieders die eten en producten aanbieden via de applicatie.

De klanten zijn weergegeven in het blauw waarbij weer een onderscheid wordt gemaakt tussen eenmalige klanten en klanten die zijn geregistreerd. Klanten die zijn geregistreerd hebben meer opties zoals het beoordelen van aanbieders, favorieten aanmaken en klantenservicegegevens inzien tot de beschikking.



Figuur 1: Usecase Diagram Homecooked

# **User stories klanten**

Op basis van de bovenstaande usecases zijn de volgende userstories voorgekomen:

1. **Inzien aanbod:** Als gebruiker wil ik door de lijst van aanbieders kunnen bladeren om te zien wat zij aanbieden en wat de recensies van deze aanbieders en hun producten is, zodat ik een overwogen keuze kan maken.
2. **Bestellen van eten**: Als gebruiker wil ik voedsel kunnen bestellen van de aanbieders via de mobiele applicatie, zodat ik snel en eenvoudig het gewenste voedsel kan ontvangen of ophalen.
3. **Bestelling aanpassen:** Als gebruiker wil ik mijn bestellingen kunnen aanpassen zoals de samenstelling van het gerecht, zodat ik de smaak en pittigheid kan aanpassen aan mijn eigen wensen.
4. **Betaalopties kiezen:** Als gebruiker wil ik bij het bestellen uit verschillende betaalopties kunnen kiezen, zodat ik altijd de goede betaaloptie beschikbaar heb.
5. **Bestelling volgen:** Als klant wil ik mijn bestelling in real-time kunnen volgen, zodat ik weet wanner mijn bestelling zal aankomen en ik mij hierop kan instellen.
6. **Inloggen als klant:** Als frequente gebruiker wil ik kunnen inloggen met mijn email of met een sociale media-account, zodat ik mijn opgeslagen restaurants/aanbieders, bestelhistorie, betaalinformatie en persoonlijke gegevens kan inzien en wijzigen.
7. **Aanbieders beoordelen:** Als klant wil ik de aanbieders en hun gerechten individueel kunnen beoordelen, zodat ik mijn ervaring kan delen met andere gebruikers en de aanbieder, die vervolgens een overwegen keuze kunnen maken.
8. **Favorieten aanmaken:** Als vaste klant wil ik mijn favoriete restaurants en gerechten combinatie kunnen opslaan, zodat ik in de toekomst sneller mijn bestelling kan plaatsen.
9. **Aanbiedingen inzien:** Als gebruiker wil ik de lopende aanbiedingen kunnen inzien, zodat ik mijn keuze aan de hand van de aanbiedingen kan aanpassen en geld bespaar.
10. **Klantenservice contacteren:** Als gebruiker wil ik contact kunnen opnemen met de klantenservice, zodat ik hulp ontvang bij eventuele vragen of misverstanden.

# **User stories aanbieders**

1. **Producten plaatsen:** Als gebruiker wil ik aanpassingen kunnen maken via de app voor de beschrijvingen, de prijzen, de afbeeldingen, het assortiment en de informatie van mijn restaurant.
2. **Bestellingen beheren:** Als gebruiker wil ik de bestellingen ontvangen en kunnen beheren, zodat ik mijn klanten goed kan bedienen.
3. **Klantbeoordelingen inzien:** Als gebruiker wil ik mijn klantbeoordelingen kunnen beheren, zodat ik kan inzien wat de beoordelingen zijn en eventueel kan reageren op de beoordelingen.
4. **Aanbiedingen plaatsen:** Als gebruiker wil ik aanbiedingen kunnen plaatsen en beheren, zodat ik gebruikers kan aansporen om een bestelling te plaatsen.

# **Vereisten**

**Functionele eisen:**

Inzien van aanbod

* De gebruiker moet in een lijst alle beschikbare aanbieders kunnen inzien. Op deze lijst moet de naam, het aantal reviews, de locatie, de afstand, de soort keuken en een afbeelding van de aanbieder te zien zijn.
* De Gebruiker moet de mogelijk hebben om te zoeken naar naam tussen de aanbieders.
* De gebruiker moet door op de aanbieder te drukken een lijst van alle mogelijke producten die deze aanbieder heeft kunnen inzien. In de lijst is de naam en de prijs van de producten zichtbaar.

Bestellen van eten

* De gebruiker moet de verschillende gerechten kunnen inzien en toevoegen aan zijn winkelwagen.
* De gebruiker moet zijn winkelwagen kunnen inzien, hierop moeten de producten het aantal, de prijs en de totaalprijs van de producten zichtbaar zijn.
* De gebruiker moet in zijn winkelwagen het aantal producten kunnen wijzigen.

Inloggen

* De gebruiker moet bij het opstarten van de applicatie de keuzemogelijkheid kunnen hebben om in te loggen of een account aan te maken middels zijn email.
* De gebruiker moet middels zijn email en een wachtwoord kunnen inloggen in de applicatie.
* De gebruiker moet middels de biometrische mogelijkheden die zijn apparaat ondersteund kunnen inloggen.

Betalen

* De gebruiker moet bij het bestellen van de gerechten de keuzemogelijkheid hebben om een betaaloptie te kiezen. (PayPal, Klarna, IDEAL, Creditcard).
* De Gebruikers moeten middels een knop/vink de mogelijkheid hebben om de betaalgegevens op te slaan voor een volgend gebruik.

Bestellen

* De gebruiker moet de producten uit zijn winkelwagen kunnen bestellen en daarbij een keuze kunnen maken tussen bezorgen en afhalen. Tevens moet er bij iedere keuze een schatting van de af te halen tijd dan wel de bezorgtijd vermeld staan.
* Na het bestellen moet de gebruiker in het overzicht van de bestellingen de bestelling kunnen volgen, waarbij de afstand of het aantal geschatte minuten worden weergegeven.
* De gebruiker kunnen in het besteloverzicht de locatie van de bezorger volgen.
* De gebruiker moet na het plaatsen van een bestelling de vraag krijgen of hij of zij een review wilt plaatsen.

Account

* De gebruiker moet zijn accountgegevens op een accountpagina kunnen inzien. Hier moeten de avatar, de persoonsgegevens, betaalgegevens en adresgegeven zichtbaar zijn.
* De gebruiker moet in staat zijn al zijn gegevens te kunnen wijzigen in het accountscherm.
* De gebruiker moet kunnen uitloggen, zijn biometrische inlogmethode aanzetten en zijn account kunnen verwijderen.

Beoordelen

* De gebruiker moet in de lijst van aanbieders op reviews kunnen drukken om alle reviews van de aanbieders te kunnen inzien.
* De gebruikers moeten hun eigen beoordelingen kunnen inzien en kunnen verwijderen.

Overige

* De gebruikers moeten op de detailweergave van de aanbieders de actuele aanbiedingen kunnen inzien.
* De gebruikers moeten via de applicatie alle klantenservice gerelateerde gegevens kunnen inzien.
* Gebruikers moeten aanbieders en gerechten als favorieten kunnen aanmerken en deze in het favorieten scherm kunnen selecteren. Ook moeten de favorieten zichtbaar zijn in de lijst van aanbieders.

Aanbieders

* Aanbieders kunnen aanbiedingen plaatsen op hun assortiment.
* Aanbieders kunnen hun lopende aanbiedingen beheren.
* Aanbieders kunnen gebruikerservaringen inzien en beheren.
* Aanbieders kunnen reageren op gebruikerservaringen.
* Aanbieders kunnen de geplaatste bestellingen beheren.
* Aanbieders kunnen hun assortiment van producten plaatsen.
* Aanbieders kunnen de soort aanpassingen aangeven per product.

**Niet-functionele eisen:**

Voor de niet functionele eisen is een onderscheid gemaakt tussen de benodigde eisen voor de Homecooked applicatie en eisen die ook voor de POC’s zal gelden. Hierbij wordt eerst een volledige lijst met eisen weergegeven en hierna de lijst die specifiek voor de POC’s van toepassing is. Dit verschil is te verklaren omdat de POC’s niet alle functionaliteit van de uieindelijke applicatie zullen bevatten.

* De app moet de gebruiker token voor het valideren van de gebruiker om de veilig native oplossing van het systeem opslaan. Deze token moet een geldigheid hebben van 6 maanden.
* De lijst van aanbieders moet binnen 1 seconden doch uiterlijk binnen 3 seconden worden getoond aan de gebruikers uitgaand van een 4G verbindingssnelheid, indien niet binnen 1 seconden de lijst verschijnt moet de gebruiker een laadscherm te zien krijgen.
* Bij het scrollen tussen aanbieders mag het aantal frames niet zakken onder de 50FPS. Uitgaand van een apparaat dat in de middenklasse valt. (vanaf 550 euro, 2023 norm).
* De gebruiker moet op het hoofdscherm kunnen schakelen tussen accountgegevens, aanbieders en winkelwagen.
* De account- en betaalgegevens van de gebruiker moeten alleen versleuteld middels een hashing algoritme worden opgeslagen. Dit hashing algoritme moet voldoen aan de ‘industry standard’.
* De status van een bestelling mag niet ouder zijn dan 5 minuten.
* De klantenservicegegevens mogen niet ouder zijn dan 2 dagen.
* De app moet 10000 gebruikers tegelijkertijd aan kunnen zonder dat er vertraging bij de gebruikers zichtbaar is.
* Bij het wisselen van applicaties moeten de app onthouden waar gebruikers zijn gebleven.
* De app moet compatibel zijn met toestellen waarbij het besturingssysteem niet ouder is dan 6 jaar.
* De app mag niet crashen, en indien dit wel gebeurt mag de gebruiker hier geen hinder van ervaren door een vastlopende telefoon.
* De app moet een consistente gebruikerservaring bieden op de verschillende appraten. Hierbij moeten de themakleuren altijd gelijk blijven.
* De applicatie moet voldoen aan de WCAG (EN 301 549) richtlijnen.
* De applicatie moet ondersteuning bieden voor Engels, Duits, Frans en Nederlands.
* De applicatie moet compatible zijn voor de volgende schermformaten (3.5inch tot 6.5inch)
* De applicatie moet voldoen aan de AVG gegevens bescherming.
* Zowel portret- als landschapsmodus moeten compatible zijn en mogen geen afwijkingen vertonen.
* De app moet relevante pushmeldingen kunnen versturen. Hierbij mag het aantal meldingen niet hoger zijn dan 1 per 5 minuut en indien het reclame gerelateerde meldingen betreft dan zijn die beperkt tot 1 per dag.

Voor het onderzoek worden een tweetal POC’s gemaakt. De niet-functionele eisen zijn aangepast aan de functionaliteit van de POC’s. Hiervoor zijn aspecten meegenomen die daadwerkelijk kunnen worden getest met de POC’s. De volgende niet-functionele eisen zijn van toepassing op de POC’s

* De lijst van aanbieders moet binnen 1 seconden doch uiterlijk binnen 3 seconden worden getoond aan de gebruikers uitgaand van een 4G verbindingssnelheid, indien niet binnen 1 seconden de lijst verschijnt moet de gebruiker een laadscherm te zien krijgen.
* De app mag niet crashen, en indien dit wel gebeurt mag de gebruiker hier geen hinder van ervaren door een vastlopende telefoon.
* De app moet compatibel zijn met toestellen waarbij het besturingssysteem niet ouder is dan 6 jaar.
* De gebruiker moet op het hoofdscherm kunnen schakelen tussen accountgegevens, aanbieders en winkelwagen. (verplaatsen naar functionele)
* Bij het wisselen van applicaties moeten de app onthouden waar gebruikers zijn gebleven.
* Zowel portret- als landschapsmodus moeten compatible zijn en mogen geen afwijkingen vertonen.
* Bij het scrollen tussen aanbieders mag het aantal frames niet zakken onder de 50FPS. Uitgaand van een apparaat dat in de middenklasse valt. (vanaf 550 euro, 2023 norm).

## **C4 Model**

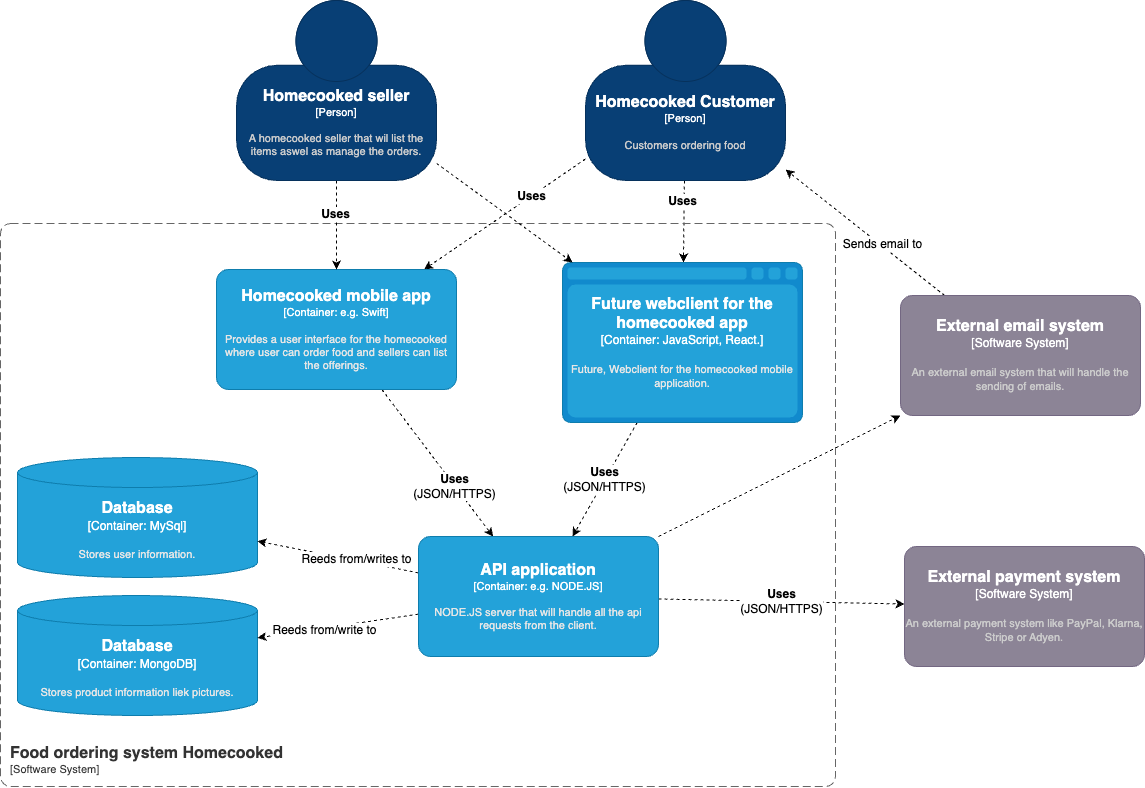
Onderstaand is de Homecooked architectuur visueel weergegeven middels het C4 model. Hierbij is een onderscheid gemaakt tussen de klanten en de aanbieders. Tevens is een webclient toegevoegd als toekomstige project.

In het C4 model is te zien hoe de mobiele client applicatie middels een API-server communiceert met de twee databases. Voor de API-server is gekozen voor Express.js dat draait op NODE.js. Door zijn minimalisme en flexibiliteit is het zeer geschikt voor een startup. Frameworks zoals Kraken, Feathers, NestJs, Sails en Locomotive maken allemaal gebruik van Express.js (IBM, Stongloop, 2023).

Express.js is een populair framework wat gebruik maakt van MVC-architectuur met een actieve community (PTY Ltd, 2023). De architectuur helpt bij het structureren en organiseren van de applicatie. Express.js heeft daarnaast ook ingebouwde routing, middleware ondersteuning, en heeft heel weinig overhead waardoor het een zeer snelle prestatie heeft.

Gezien de performance is voor het opslaan van productgegevens gekozen voor een NoSQL database. Voor de POC’s wordt omtrent tijd voor alle gegevens gebruik gemaakt van de NoSQL MongoDB database.

Het verzenden van e-mails alsmede het afhandelen van betalingen wordt afgehandeld via externe systemen die hierop zijn gespecialiseerd. Hierdoor wordt een stuk abstractie toegepast en wordt de kans op fouten alsmede de tijd tot aan een MVP(minimum viable product) verkleind.

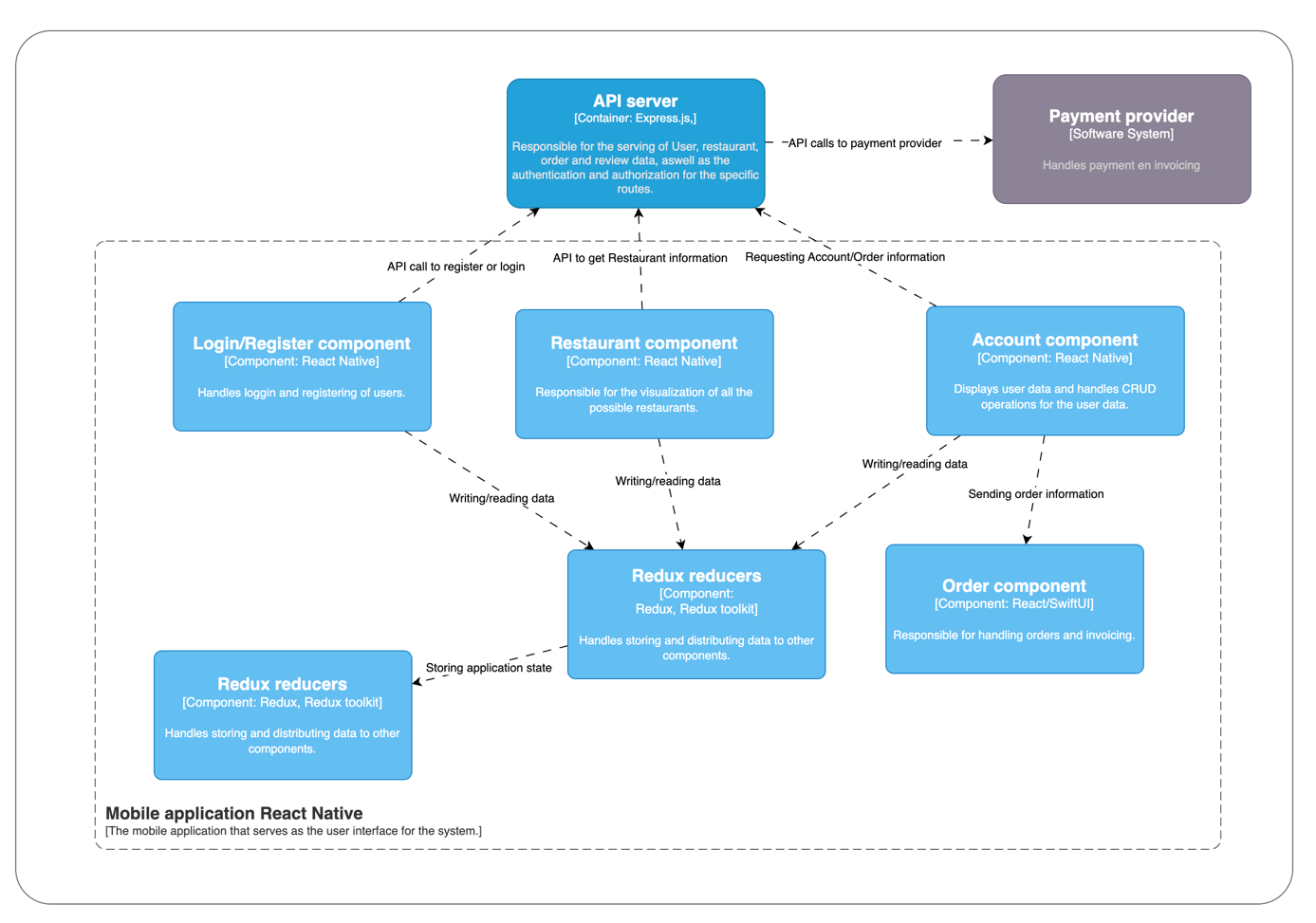


Figuur 2: C4 model niveau 2 System level

De Raect Native mobiele front-end applicatie heeft een drietal componenten die communiceren met de API server. Deze opsplitsing zorgt ervoor dat de taken worden verdeeld en dat het mogelijk is om bijvoorbeeld restaurant gegevens in te zien zonder in te loggen.

Vervolgens wordt middels het statemanagementsysteem Redux de afhandeling van de gegevens verwerkt. Redux is een veel gebruikte manier van state management die vooral goed past binnen de applicatie. Immers moet er redelijk veel state op applicatie niveau worden bijgehouden.

De overige componenten zijn verdeeld naar taken die zij afhandelen. Hierdoor wordt als het ware seperation of concerns toegepast. Hoewel dit voor een front-end niet noodzakelijk is, aangezien er niet veel business logica zit, vereenvoudigt het wel de applicatie.



Figuur 3 C4 component diagram React Native applicatie

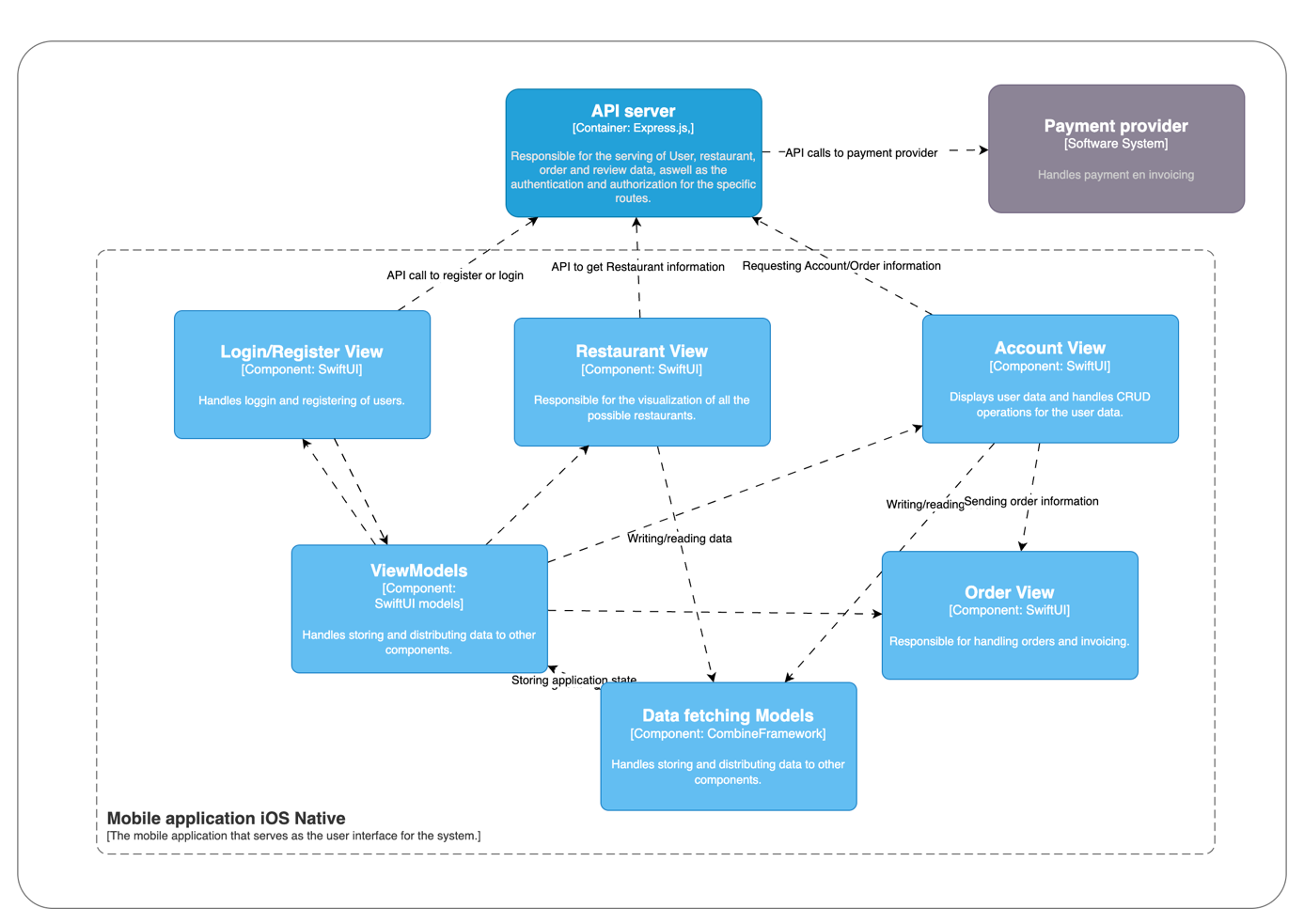
De Native iOS mobiele front-end applicatie verschilt een beetje van de react native applicatie. Hierbij zijn de verschillende componenten ondervwerdeel in ‘Views’.

tegenstelling tot de React Native applicatie wordt het statemanagement grotendeels door het SwiftUI raamwerk afgevangen. De ontwikkelaars worden hierbij heel veel handvatten toegereikt die aanbevalen worden door Apple. Hierdoor is het verstandig om deze handvatten ook zoveel mogelijk te gebruiken.

Voor het fetchen van data wordt een relatief nieuw raamwerk van Apple gebruikt, namelijk het Combine raamwerk.

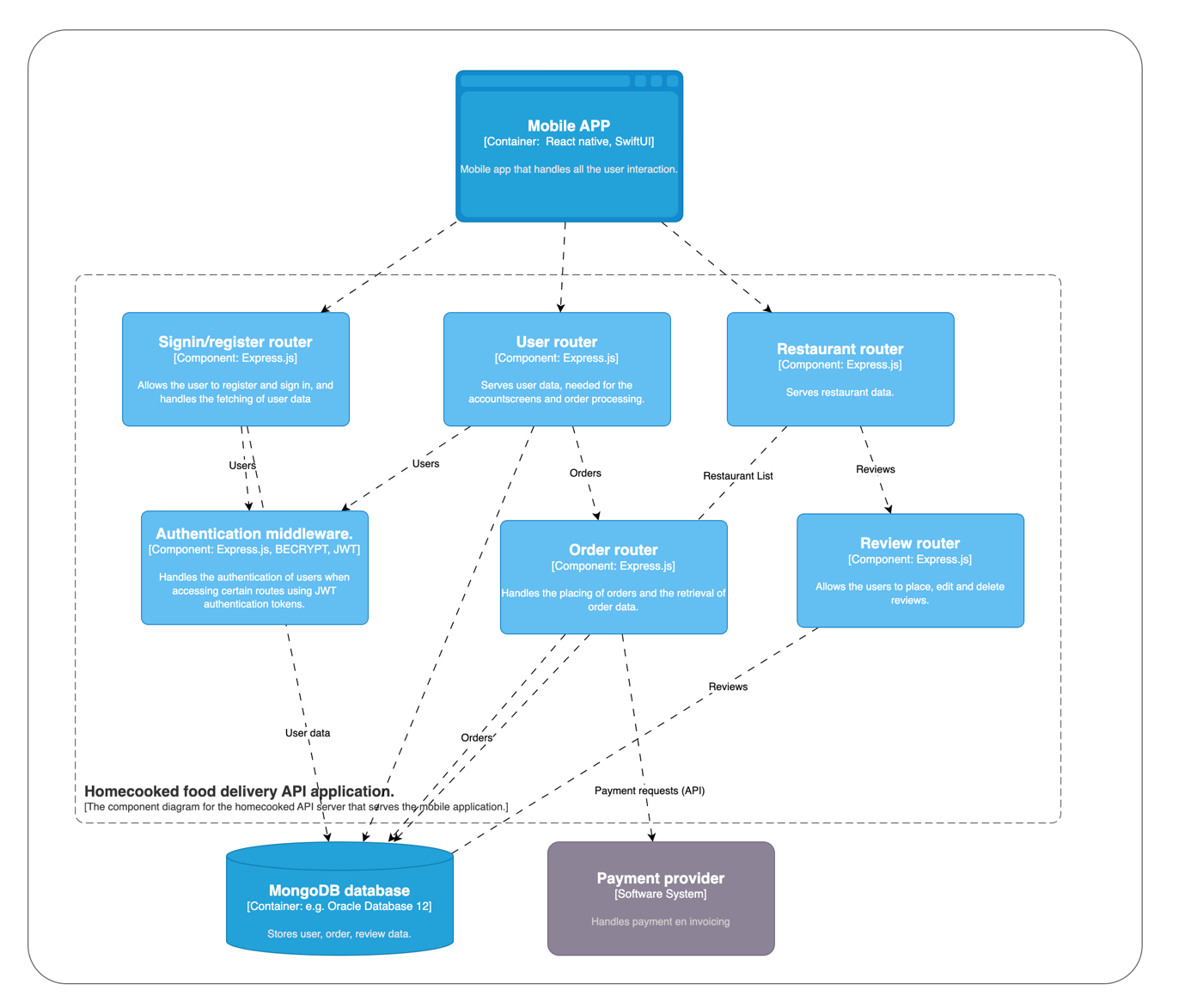
Aangezien Swift gebruik maakt van ‘types’ is er een globaal component waarin alle modellen worden aangemaakt die door de applicatie wordt gebruikt.

Figuur 4 C4 component diagram iOS Native applicatie



In het component level diagram van de API-server zijn wederom de verschillende communicatie lijnen opgesplitst en corresponderen zij met de zelfde componenten die bij de front-end worden gebruikt. Dit is gedaan omwille de eenvoud en de splitsing van bevoegdheden en taken.

Voor de authenticatie en autorisatie is gebruik gemaakt van een middleware die aan de verschillende routes kan worden gehangen. De wachtwoorden van gebruikers worden middels Bcrypt encryptie versleuteld opengeslagen. Waarbij een Salt aan de wachtwoorden wordt toegevoegd voor het minimaliseren van bijvoorbeeld Rainbow Table aanvallen (Wikipedia, sd).



Figuur 5 C4 component diagram API-server

# **Welk ontwikkelraamwerk is geschikt voor het ontwikkelen van de mobiele applicatie rekening houdend met de vooraf gedefinieerde randvoorwaarden?**

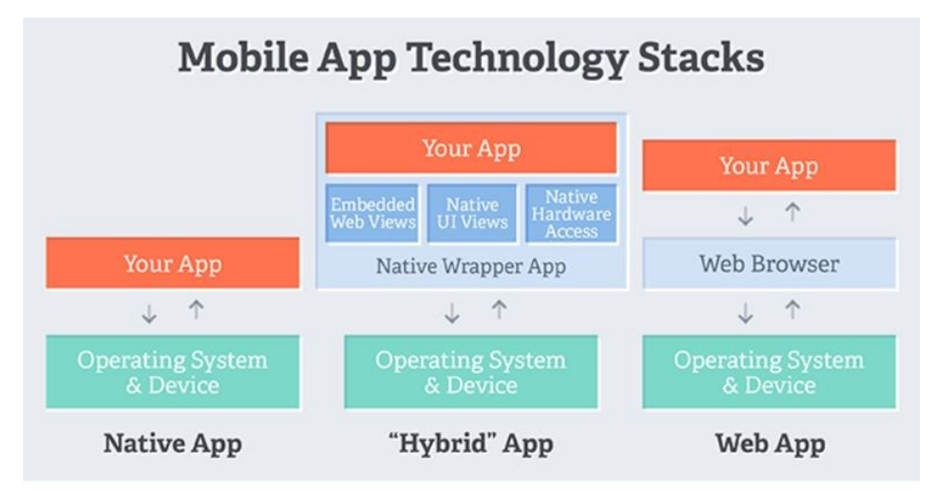
Voor het bepalen van een geschikt raamwerk is het belangrijk alle aspecten en eisen mee te nemen in de bepaling hiervan. Hiervoor is eerst een lijst gemaakt met de beschikbare mogelijkheden waarmee een mobiele applicatie kan worden ontwikkeld. Vervolgens moet worden gekeken naar de sterkte en zwaktes van de verschillende raamwerken. Hierna kunnen dan ook aspecten als tooling en de ontwikkelomgeving voor ontwikkelaars. Immers staat de ontwikkelervaring van ieder raamwerk haaks aan het vinden van ontwikkelaars die daadwerkelijk ermee aan de slag willen gaan.

# **Raamwerken overzicht**

Mobiele ontwikkeling is tegenwoordig vaak onmisbaar bij het aantrekken van gebruikers of bij het implementeren van de nieuwste functionaliteiten of mogelijkheden. Onder mobiele ontwikkeling wordt vaak aan mobiele telefoons gedacht, echter gaat mobiele ontwikkeling verder dan mobiele telefoons. Hierbij kunnen ook bijvoorbeeld smartwatches, smart armbanden, tablets, convertables en vouwschermen onder vallen. De scope van dit onderzoek zal zich beperken tot mobiele telefoons die draaien op de twee grootste besturingssystemen Android en iOS.

Voor het ontwikkelen van mobiele applicaties zijn er verschillende raamwerken of technologieën beschikbaar.

Ieder van deze technologieën geeft zijn voor en nadelen. Voor het bepalen van het raamwerk of technologie zijn de mogelijkheden onderverdeeld in 4 categorieën.



Figuur 5 Overzicht technology stack raamwerken

### **Native app ontwikkeling**

Ten eerste is het mogelijk om “Native” applicaties te ontwikkelen. Hiermee worden de applicaties bedoeld die zijn ontwikkeld met de software die de makers van de twee populairste besturingssystemen hebben meegeleverd voor de ontwikkeling van de applicaties (Microsoft, sd).

Voordelen:

* Native applicaties zijn geoptimaliseerd en bieden een beter prestatie en gebruikerservaring (Schmitt, sd)
* Kunnen gebruik maken van alle native API’s en functies van het besturingssysteem
* Geen dataverbinding nodig

Nadelen:

* Het ontwikkelen vergt expertise van verschillende programmeertalen en ontwikkelomgevingen om voor beide platformen te ontwikkelen.
* Onderhouden van meerdere codebases is complex en tijdrovend.

Applicaties die native ontwikkeld zijn:

* Whatsapp
* Spotify
* Pokemon Go
* Waze

### **Cross platform native applicaties**

Cross platform applicaties zijn ontworpen om op verschillende besturingssystemen en apparaten te werken met een enkele codebase. Verder worden voor verschillende frameworks native elementen gebruikt om de user interface op te bouwen en om een zoveel mogelijke native ervaring te geven (Microsoft, sd) (Mosconi, 2022).

Voordelen:

* Cross-platform applicaties kunnen worden ontwikkeld met een enkele codebase.
* Kunnen op verschillende platforms worden ingezet.

Nadelen:

* Prestatie van cross-platform apps komt niet overeen met die van native applicaties.

Applicaties die crossplatform zijn ontwikkeld (Flutter, sd) (Meta, sd):

* Facebook (iOS, Android) - React Native
* Microsoft Teams - React Native
* Outlook - React Native
* Tesla - React Native
* PS app - React Native
* Discord Bloomberg - React Native
* BMW app - Flutter
* Toyota app - Flutter
* Google pay - Flutter
* Nubank - Flutter

### **Hybride mobiele applicaties**

Hybride applicaties zijn zoals crossplatform applicaties geschreven met een code base die werkt op verschillende besturingssystemen en apparaten. Het voornaamste verschil tussen hybride en crossplatform is de opbouw van de gebruikersinterface. Waarbij de gebruikersinterface van hybride apps is geschreven met web technologieën.

Voordelen:

* Hybride apps kunnen op meerdere platforms worden ingezet.
* Native functies en API’s kunnen middels plugins worden benaderd.

Nadelen:

* Prestatie van Hybride apps komt niet overeen met die van native applicaties.

Applicaties die hybride zijn:

* Bobcat - Ionic
* Southwest airlines - Ionic

### **Progressive web apps (PWA)**

“Progressive web apps” zijn webapplicaties ontwikkeld om te optimaal te kunnen worden gebruikt op mobiele apparatuur. De applicatie wordt altijd geopend door een webbrowser zonder dat daarvoor installatie nodig is. Hierbij worden geen afzonderlijke versies voor de verschillende besturingssystemen gebruikt.

Voordelen:

* Kunnen rechtstreeks vanuit het web worden geopend waardoor geen installatie nodig is.
* Kunnen op verschillende platforms worden gebruikt.

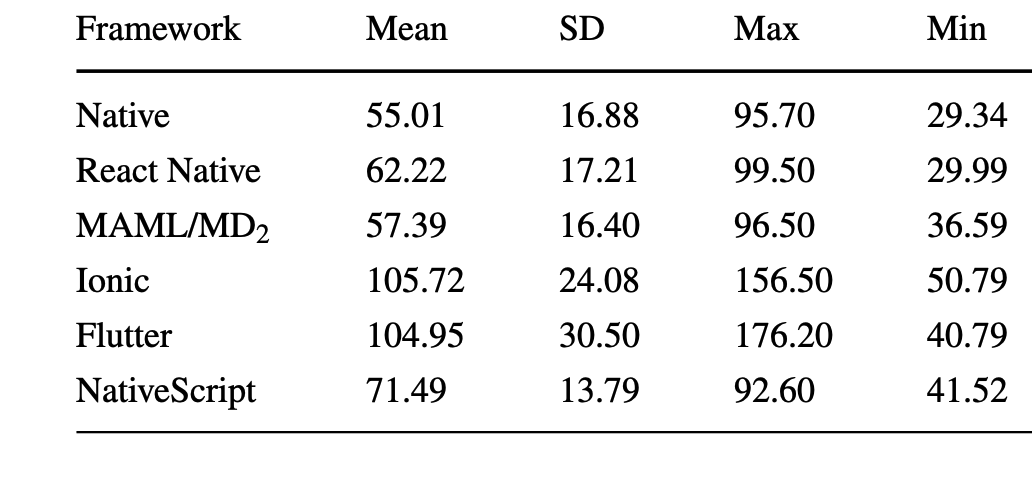
Nadelen:

* Hebben beperkte toegang tot native API’s en functies.
* Prestatie komt niet overeen met zowel native, hybride als crossplatform applicaties.

## **Memory gebruik**

Het memorygebruik van een applicatie kan een relevante factor zijn bij het bepalen van de prestatie van een framework. Door het gebruik van niet native oplossingen in de verschillende frameworks kan het RAM gebruik anders uitvallen.

In het onderstaande overzicht is het memorygebruik van een applicatie die op ieder onderzocht framework op de zelfde wijze is nagebouwd onderzocht (Biorn-Hansen, Rieger, Gronli, Majchrzak, & Ghinea, 2020). Hierbij is voor het maken van de native applicatie een Android native applicatie gebouwd. Hierbij is een onderscheid gemaakt tussen RAM-gebruik tijdens de “IDLE-state” van de applicatie en tijdens intensief gebruik.



Tabel : RAM-gebruik intensief gebruik (Biorn-Hansen, Rieger, Gronli, Majchrzak, & Ghinea, 2020)

Tabel RAM-gebruik ‘IDLE-state’ (Biorn-Hansen, Rieger, Gronli, Majchrzak, & Ghinea, 2020)

Uit het overzicht is te zien dat de Native oplossing onder de applicaties met vergelijkbare functionaliteit en prestatie wat betreft de UI consistentie het laagste RAM-gebruik ondervindt. Hierbij zijn een aantal grote uitblinkers zoals Flutter en Ionic te onderscheiden die beduidend meer RAM-gebruik vertoond. Hoger ram-gebruik zou kunnen leiden tot vertraging op mobiele appraten die over weinig ram beschikken. Ook leidt hoger RAM-gebruik tot meer energieverbruik.

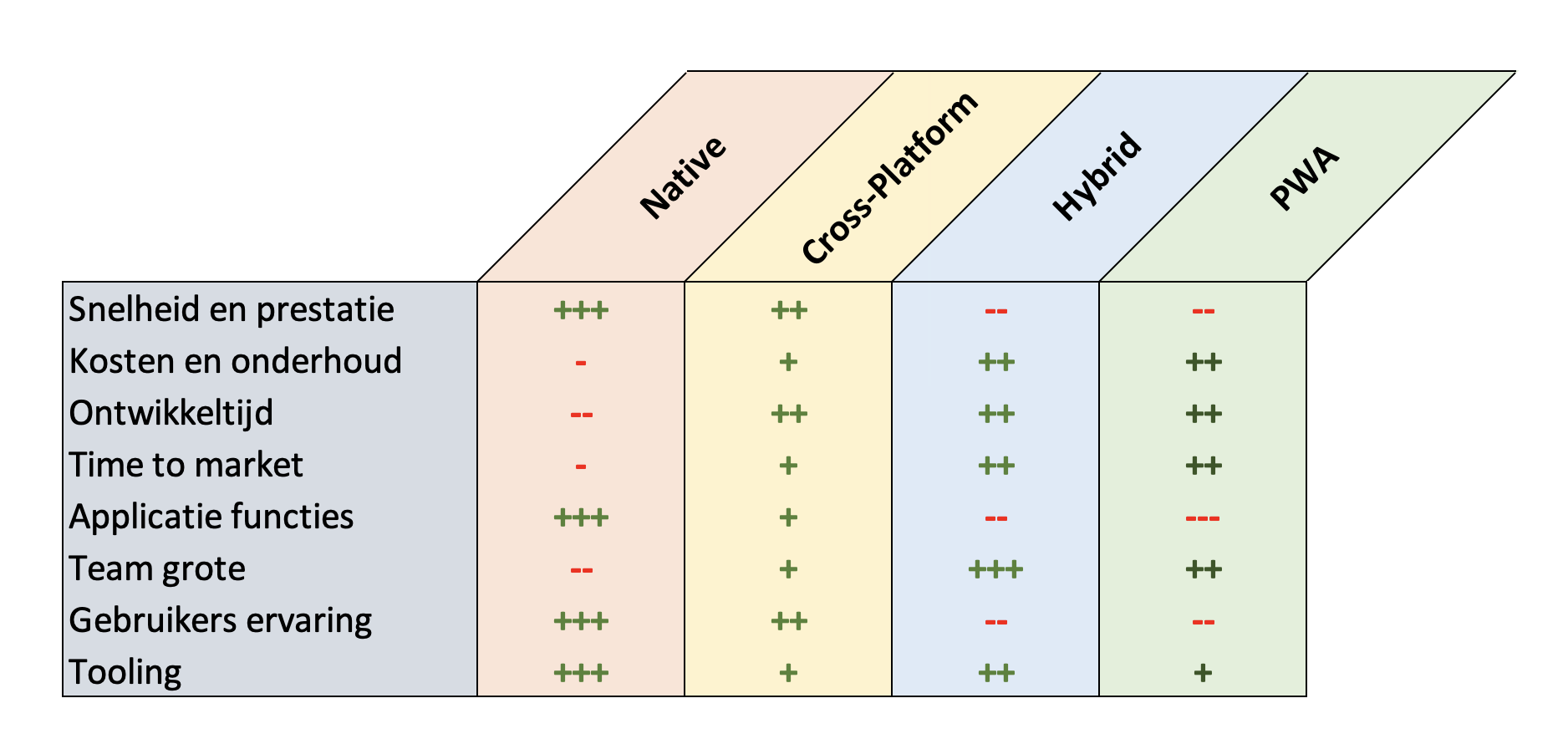
## **Overwegingen Homecooked**

Voor het bepalen van een geschikt raamwerk voor het ontwikkelen van een mobiele applicatie zijn er een aantal factoren die moeten worden overwogen. Onderstaand overzicht geeft de lijst aan aspecten waarop de verschillende raamwerken worden beoordeeld. Gezien de beperkte beschikbaarheid van tijd en middelen wordt aan de hand van deze aspecten een keuze gemaakt voor twee raamwerken waarin de Proof of Concepts nader worden uitgewerkt.

* Snelheid en prestatie: Hiermee wordt de prestatie van de applicatie bedoeld, zoals de snelheid van openen van de applicaties, het inzien van data en de framesnelheid die de “smootheness” bepaald.
* Kosten en onderhoud: De kosten voor het ontwikkelen en onderhouden van een applicatie spelen een belangrijk rol in de keuze van het raamwerk. Hierbij wordt een inschatting gegeven van de totale kosten voor de ontwikkeling en onderhoud.
* Ontwikkeltijd: De ontwikkeltijd wordt gemeten in manuren die nodig zijn om de applicatie als MVP ten ontwikkelen en uit te brengen.
* Time to market: De time to market geeft de tijd van ontwerp tot deployment van de applicatie weer op de twee ondersteunende platforms.
* Applicatie functies: De gewenste functionaliteit speelt een belangrijke rol bij de keuze van het ontwikkelplatform. Bepaalde functies zijn alleen beschikbaar bij een native oplossing waarbij andere vormen afvallen.
* Team grote: De grote van het team heeft een sleutelrol in het bepalen van bijvoorbeeld de time to market of de ontwikkelkosten. Er is immers aan schaarste aan ontwikkelaars.
* Gebruikerservaring: De gebruikerservaring speelt een zeer belangrijke rol in de bepaling van een raamwerk. Deze ervaring staat ook in verband met de snelheid en prestatie van de applicatie. Tevens kunnen raamwerken specifieke functies bevatten die deze ervaring verbeteren. De gebruikerservaring is tevens een belangrijke factor voor het succes van een applicatie. Native applicaties kunnen de beste gebruikerservaring bieden, terwijl native en hybride applicaties wellicht concessies moeten maken op verschillende aspecten.
* Tooling: Met tooling wordt de ontwikkelomgeving en alle support rondom het ontwikkelen van de applicatie bedoeld. De ontwikkelaarservaring speelt hierbij een grote rol. Tevens is de beschikbaarheid van libraries en raamwerken voor het testen en deployen van applicaties een belangrijk onderdeel hiervan.

## **Keuze POC’s**

In het onderstaande overzicht is een uiteenzetting van de verschillende raamwerken gemaakt. Waarbij er een puntensysteem is gehanteerd voor de verschillende onderdelen. Zo zijn een aantal onderdelen zoals de gebruikers- en ontwikkelaarservaring zeer belangrijk voor de keuze van een raamwerk.



Tabel Overzicht SWOT analyse raamwerken

Op basis van bovenstaande lijst is de keuze voor het ontwikkelen van de POC’s gevallen op een Native ontwikkelde en Cross-platform ontwikkelde applicatie. De keuze hiervoor is voor een vooral toe te wijzen aan de gebruiksvriendelijkheid, ontwikkeltijd en de tooling van de twee raamwerken.

De applicatie heeft eis dat de gebruiksvriendelijkheid bovenaan staat. Immers is de eerste indruk die je van een applicatie krijgt zeer belangrijk voor het verkrijgen van klanten. Zo is een van de top 5 grootste valkuilen van een slecht presterende app de snelheid hiervan (Top 5 User Experience Issues In Mobile Apps, 2021).

## **POC testcases**

Op basis van de ontwikkelde POC’s en de vooraf gedefinieerde niet-functionele eisen zijn een aantal testcases ontwikkeld. Ieder POC’s zal op basis van de testcases worden getest.

De testcases zijn onder te verdelen in twee categorieen. Enerzijds zijn er testcases geschreven voor het prestatie aspect van de POC’s en anderzijds zijn er binaire testcases die aangeven of bepaalde functionaliteit wel of niet wordt ondersteund.

De prestatie gerelateerde testcases zijn als volgt:

* Frame per seconde bij niets doen
* RAM gebruik bij niets doen
* RAM gebruik nadat alle tests zijn afgerond
* Het aantal “dropped” FPS
* De FPS prestatie van de lijst met verschillende grootten(10, 100, 1000, 10000)
* Laadsnelheid van de lijst met restaurants

Voor de tweede categorie is gekeken naar de volgende aspecten:

* Compatibiliteit voor oudere apparaten
* Compatibiliteit voor verschillende schermgrootten
* Compatibiliteit voor zowel landschap- als portret-modus

## **POC testresultaten**

Op basis van de testresultaten is duidelijk naar voren gekomen dat het aantal FPS voor de native iOS applicatie het meest betrouwbare aantal laat zien. Hierbij zitten cross-platform varianten niet ver van de native variant af. De verschillen zullen derhalve ook voor de alledaagse gebruiker niet te onderscheiden zijn.

Tabel Overzicht FPS per POC

Voor het RAM-gebruik is er wel een groot verschil te onderscheiden. Hierbij zijn de crossplatform applicaties aanzienlijk groter bij ‘idle’ gebruik. Ook is de toename van het RAM tijdens het gebruik van de applicatie aanzienlijk groter dan bij de cross-platform varianten. Dit verschil is enerzijds te verklaren omdat er allerlei native libraries moeten worden gedownload bij de cross-platform varianten. Tevens hebben de cross-platform varianten een bepaalde overhead voor het hanteren van de JavaScript thread.

Het RAM-gebruik zal bij nieuwere telefoons met veel RAM geen probleem zijn. Echter is het zo dat oude en goedkope android telefoons weinig RAM hebben en dit zou problemen kunnen veroorzaken. Tevens is het zo dat oudere iPhones ook relatiev minder RAM ter beschikking hebben.

Tabel Overzicht RAM gebruik per POC

Alle drie de varianten hebben heen crashes veroorzaakt en ondervinden geen meetbaar stotterende UI.

Wat verder opvalt is dat de laadsnelheid van de lijst met restaurants op de native iOS variant langzamer is dan de cross-platform varianten. Dit is ook tevens het enige onderdeel in de testcases waar de iOS native applicatie minder goed presteert dan de cross-platform applicaties.

## **Ontwikkelervaring**

## **Conclusie**

## Voor het bepalen van een geschikt raamwerk zijn heel veel aspecten die invloed hebben op de keuze van het raamwerk. Voorafgaand aan dit project zijn een aantal niet-functionele eisen opgesteld waaraan de applicatie moet voldoen. Deze niet-functionele eisen zijn tevens ook de minimale eisen waaraan het raamwerk moet voldoen.

## Uit de testresultaten is naar voren gekomen dat zowel de native als de cross-platform varianten geschreven in React native voldoen de niet-functionele eisen die zijn opgesteld. Hierbij zijn een aantal kanttekeningen te ondervinden. Het hoge RAM-gebruik kan een negatieve impact hebben op de prestatie van de applicatie, met name voor apparaten die weinig RAM tot beschikking hebben. Hierbij is bij de cross-platform varianten met name ook de ‘idle ram’ gebruik zeer hoog. Dit staat dus los van de grootte van de applicatie. Het startpunt is dus dusdanig hoog dat optimaliseren van de applicatie weinig zal uitbrengen om dit startpunt te verlagen.

## Naast de niet-functionele eisen moet er ook worden gekeken naar de ontwikkeltijd en de middelen die moeten worden ingezet. Tevens is bij een startende onderneming de time to market zeer belangrijk. Bij een native applicatie moeten dus twee native applicaties worden ontwikkeld om de grootste aantal potentiële klanten te benaderen. Hierbij moeten dus meer middelen en meer manuren worden gebruikt voor het ontwikkelen van de applicatie. Daarnaast is nog maar de vraag of de kennis in huis is om de native oplossingen te ontwikkelen.

Verder zijn alle benodigde mobiele functies die zullen worden gebruikt aanwezig bij de cross-platform varianten.

Met inachtneming van het voorgaande is voor dit project de keuze gevallen op het ontwikkelen met de cross-platform varianten. Hierbij zijn vooral de beschikbare kennis, de teamgrootte, de middelen en de ontwikkelervaring de doorslaggevende factoren geweest in de bepaling van het raamwerk.

Aangezien de middelen zeer beperkt zijn en het een startende onderneming betreft moeten de middelen zo efficiënt mogelijk worden ingezet. Hierbij is de keuze om een groter publiek te benaderen of een iets betere prestatie van de applicatie relatief eenvoudig geweest.

Daarnaast is de ontwikkelervaring met de POC van de native applicatie dusdanig verschillend en anders in negatieve zin dat ook dit onderdeel relatief zwaar weegt bij de keuze van het raamwerk.

## **Kanttekeningen**

Voor de keuze van het raamwerk zijn een aantal aspecten voor de toekomst waar verder onderzoek naar kan worden gedaan.

De ontwikkelde POC’s die zijn gebruikt voor de testcases geven natuurlijk geen volledig beeld over de prestaties van een volwaardige applicatie. Hierbij kan het zo zijn dat bij een grootere applicatie de prestatie tegenvallen en verder uiteen zitten.

Verder is bij de keuze van mobiele functionaliteit uitgegaan van de stand van zaken op moment van het schrijven het onderzoek. Hierbij zijn alle vereiste functies op dit moment ook aanwezig en worden deze ook ondersteund bij de cross-platform varianten.

De keuze tussen native en cross-platform varianten is in dit geval specifiek toegepast op het React native raamwerk. Andere cross-platform raamwerken kunnen afwijkende prestaties vertonen. Hierdoor kan de keuze anders uitvallen.

# Bibliografie

Biorn-Hansen, A., Rieger, C., Gronli, T.-M., Majchrzak, T., & Ghinea, G. (2020). An empirical investigation of performance overhead in cross-platform mobile development frameworks. *Springer*.

Bohara, M. (sd). Comparison between React native and native application devcelopment platforms.

Flutter. (sd). *Flutter Showcases*. Opgehaald van Flutter: Https://flutter.dev/showcase

IBM, Stongloop. (2023). *Frameworks built on Express*. Opgehaald van Expressjs.com: https://expressjs.com/en/resources/frameworks.html

IBM, Strongloop. (2023). *Frameworks*. Opgehaald van Express.js: https://expressjs.com/en/resources/frameworks.html

Majchrzak, T., & Heitkotter, H. (sd). *Status quo and best practices of app development in regional compa- ¨ nies. I*. Opgehaald van Springer.

Meta. (sd). *React native showcases* . Opgehaald van React native : https://reactnative.dev/showcase

Microsoft. (sd). *Native, hybrid, or cross-platform apps?* Opgehaald van Microsoft: https://powerapps.microsoft.com/en-us/native-vs-cross-platform-apps/

Mosconi, N. (2022, November). *Native vs Cross-platform Development: Pros and Cons*. Opgehaald van Devlane: https://www.devlane.com/blog/cross-platform-apps-vs-native-apps-pros-cons

PTY Ltd. (2023). *Express Usage Statistics*. Opgehaald van Builtwith: https://trends.builtwith.com/framework/Express

Schmitt, J. (sd). *Cicleci*. Opgehaald van Circleci: https://circleci.com/blog/native-vs-cross-platform-mobile-dev/?psafe\_param=1&utm\_source=google&utm\_medium=sem&utm\_campaign=sem-google-dg--emea-en-dsa-tROAS-auth-nb&utm\_term=g\_-\_c\_\_dsa\_&utm\_content=&gclid=CjwKCAjwgqejBhBAEiwAuWHioMuAtyYOjAy8mgV5LCd\_x4k7oBa

*Top 5 User Experience Issues In Mobile Apps*. (2021). Opgehaald van Idealogic: https://idealogic.dev/5-top-user-experience-issues-in-mobile-apps/

Wikipedia. (sd). *Rainbow Table*. Opgehaald van Wikipedia: https://en.wikipedia.org/wiki/Rainbow\_table