Een onderzoek naar de maturiteit van .NET MAUI bij het vervangen van een real-time POS-systeem.

Michiel Van Herreweghe.

Scriptie voorgedragen tot het bekomen van de graad van
Professionele bachelor in de toegepaste informatica

Promotor: Mevr. M. Van Audenrode
Co-promotor: Dhr. K. Van Moorter
Academiejaar: 2022–2023
Eerste examenperiode

Departement IT en Digitale Innovatie .

HO
GENT

Woord vooraf

.NET MAUI is het nieuwste project van Microsoft dat ons toelaat om cross-platform applicaties te ontwikkelen met C# en .NET.
Ikzelf ben altijd graag bezig met het uitproberen van nieuwe tools, programmeertalen en features. Dit onderzoek is mij dan ook op het lijf geschreven.
Eerst en vooral wil ik mijn co-promoter **Kristof Van Moorter** bedanken. Kristof heeft mij met raad en daad bijgestaan op de momenten dat ik het nodig had. Daarnaast wil ik mijn promotoren **Martine van Audenrode** en **Chantal Teerlinck** bedanken die altijd de gepaste feedback gaven wanneer ik het vroeg. Ook wil ik heel graag mijn ouders, **Mario Van Herreweghe** en **Sabrina Timmermans**, bedanken die mij alle kansen gegeven hebben en mij gevormd hebben tot wie ik nu ben. Verder wil ik ook mijn vriendin **Ilke Kestemont** bedanken voor alle hulp en ondersteuning die zij mij geboden heeft tijdens het onderzoeken en schrijven van deze bachelorproef.
Tot slot wil ik deze bachelorproef ook opdragen aan mijn peter **Filip Timmermans**, die er niet meer is. Hij zal mij nooit zien afstuderen, maar op deze manier is hij er toch bij.

Samenvatting

.NET MAUI (Multi-platform App UI) is een nieuw framework voor de ontwikkeling van cross-platform toepassingen die ingezet kan worden voor zowel de desktop als mobiele platformen. Het is ontwikkeld door Microsoft en komt voort uit het eerdere Xamarin.Forms framework. Met NET MAUI kunnen ontwikkelaars een enkele codebase schrijven en delen op verschillende platforms, waaronder iOS, Android, Windows en macOS.
Een bedrijf dat nog steeds gebruik maakt van een oud kassassysteem geschreven in COBOL, stootte afgelopen zomer op het probleem dat er geen floppydisks meer geproduceerd worden. Voor Goossens NV, het bedrijf in kwestie, is dit een groot probleem aangezien de gegenereerde dagrapporten van alle winkels op de bovengenoemde floppydisks geplaatst worden. Daarnaast willen ze het systeem uitbreidbaar maken met oog op de toekomst. Dit alles zorgt ervoor dat een ontwikkeling van een nieuw systeem zich aandringt.
Deze proef gaat na of .NET MAUI al voldoende ontwikkeld is om een real-time kassasysteem geprogrammeerd in COBOL te vervangen door een systeem geschreven in C# gebruikmakend van het .NET MAUI-framework. De centrale onderzoeksvraag van deze proef luidt dan ook als volgt: “Is .NET MAUI al voldoende matuur om een bestaande POS-systeem geschreven in COBOL te vervangen?”.
De uitvoering van dit onderzoek heeft als doel om het volledige systeem van een copycenter te simuleren. Dit systeem zal een analoog signaal van de printers moeten kunnen opvangen, omzetten in een digitaal signaal en doorgeven aan de grafische interface van het kassasysteem. Dit alles moet real-time gebeuren, wat wil zeggen dat vanaf het moment dat een digitaal signaal uitgezonden wordt, het kassasysteem zo goed als onmiddelijk zichzelf update met de nieuwe tellerstand. Daarnaast moet het ook mogelijk zijn om push-notificaties te versturen bij het afsluiten van het syteem. Het succes van de proef kan dan ook gedefinieerd worden op basis van twee criteria. De proef kan aanschouwd worden als een partieel succes wanneer de gemiddelde reactiesnelheid van de .NET MAUI-applicatie sneller is dan de gemiddelde reactiesnelheid van het huidige systeem. De proef is een geheel succes indien het versturen van push-notificaties mogelijk is met .NET MAUI.
Om een antwoord op de centrale onderzoeksvraag te kunnen formuleren, wordt een tweedelig onderzoek uitgevoerd. Het onderzoek bestaat uit een literatuurstudie, die als doel heeft context te schetsen, en een proof-of-concept, die bedoeld is om metingen op uit te voeren.
Uit de proof-of-concept blijkt dat de gemiddelde reactiesnelheid van .NET MAUI

ongeveer drie en een half keer sneller is dan de reactiesnelheid van het huidige systeem. Daarnaast werd ook opgemerkt dat .NET MAUI geen algemene oplossing biedt om push-notificaties te versturen zonder platformspecifieke code te gebruiken.
De conclusie die getrokken kan worden uit bovenvermelde resultaten is dat deze proef een partieel succes is. Enerzijds is de gemiddelde reactiesnelheid van de .NET MAUI-applicatie sneller dan die van het huidige systeem, anderzijds is het niet mogelijk om push-notificaties te versturen zonder platformspecifieke code te gebruiken.

Inhoudsopgave

Lijst van figuren

1

Inleiding

1.1. Probleemstelling

Goossens NV, een bedrijf dat in gereviseerde printers doet, heeft ook enkele copycentra waar momenteel een zogenaamd *"Point-Of-Sale-systeem"* draait. Dit is geschreven in de COBOL-programmeertaal. Aan het einde van elke dag wordt de kassa afgesloten en wordt door het systeem steeds een dagrapport gegenereerd en opgeslagen op een floppydisk. Echter aan het begin van de zomer stootte het bedrijf op het probleem dat er geen floppydisks meer geproduceerd werden. Daarnaast is het tegenwoordig ook verplicht om elektronische betalingen mogelijk te maken. Deze zaken zorgden er dan ook voor dat er nood was aan het moderniseren van het POS-systeem.
Dit onderzoek gaat na of het .NET MAUI framework van Microsoft al voldoende matuur is om het bestaande POS-systeem geschreven in COBOL te vervangen door een .NET MAUI-applicatie.
Het .NET MAUI framework werd gekozen voor dit onderzoek omdat dit framework enkele voordelen met zich meebrengt:

1. Integratie

   .NET MAUI maakt deel uit van het .NET-ecosysteem. Dit laat toe om in de toekomst meer tools en/of applicaties te ontwikkelen die simpel te integreren zijn met het kassasysteem.

2. Gebruikersinterface definitie met HTML

   Het .NET MAUI framework laat toe om de gebruikersinterface te definiëren met twee verschillende programmeertalen: XAML en HTML. Voor ontwikkelaars die gewend zijn van webapplicaties te ontwikkelen is de stap naar .NET MAUI aanzienlijk kleiner aangezien zij geen nieuwe taal moeten leren.

3. Cross-platform

   Indien in de toekomst het kassasysteem op een mobiel platform zoals iOS of Android moet draaien, dan is dit mogelijk met een .NET MAUI applicatie. Dit framework laat toe om met één codebase applicaties te ontwikkelen voor meerdere platformen.

1.2. Onderzoeksvraag

De onderzoeksvraag voor dit onderzoek luidt als volgt: “is .NET MAUI al voldoende matuur om een bestaande POS-systeem geschreven in COBOL te vervangen?”.
Om dit te kunnen onderzoeken zijn er enkele subonderzoeksvragen die moeten leiden tot een conclusie op de hoofdvraag:

1. Wat is de gemiddelde reactiesnelheid van de .NET MAUI applicatie en hoe verhoudt deze zich ten opzichte van de gemiddelde reactiesnelheid van het huidige systeem?
2. Laat .NET MAUI het toe om push-notificatie te versturen?

1.3. Onderzoeksdoelstelling

Tijdens dit onderzoek zal een proof-of-concept opgezet worden die het volledige systeem van een copycenter simuleert. Tijdens het opzetten zal getracht worden om een python script te maken die het analoge signaal van een printer opvangt en omvormt naar een digitaal signaal met een payload. Ook zal nagegaan worden of .NET MAUI het toelaat om push-notificatie te versturen.
Daarna zullen metingen op de proof-of-concept en op het huidige systeem uitgevoerd worden. Deze zullen de gemiddelde reactiesnelheid applicaties peilen en vergelijken met elkaar.
De proef is een partieel succes indien:

- De gemiddelde reactiesnelheid van de .NET MAUI-applicatie sneller is dan de gemiddelde reactiesnelheid van het huidige POS-systeem.

De gehele proef is een succes indien:

- Het versturen van push-notificaties mogelijk is met .NET MAUI.

1.4. Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:
In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomein, op basis van een literatuurstudie.
In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.

In Hoofdstuk 4 wordt de opzet van de proof-of-concept besproken en uitgelegd welke metingen op deze proof-of-concept uitgevoerd zijn.
In Hoofdstuk 5, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2

Stand van zaken

Inleiding

In dit hoofdstuk worden een aantal zaken besproken met het oog op de contextualisering van de studie. Allereerst zal kort ingegaan worden op de programmeertaal COBOL. Daarna wordt het verschil belicht tussen native en cross-platform ontwikkeling, elk met zijn eigen voor- en nadelen. Nadien volgt een bespreking van het .NET MAUI framework. Tenslotte zal ook het onderwerp real-time toepassingen worden toegelicht.

2.1. COBOL

2.1.1. Een korte geschiedenis van COBOL

In de jaren vijftig was de informatica vooral gericht op de rekenkracht van computers, die voornamelijk in de wiskunde en de wetenschap kon worden gebruikt. Financiële instellingen zagen echter ook de voordelen van computertoepassingen in het bedrijfsleven. In 1959 pleitte Mary Haws voor de ontwikkeling van een programmeertaal die kon worden gebruikt voor zakelijke doeleinden, zoals salarisadministratie, inventarisatie en analyse van bedrijfsgegevens (National Museum of American History, 2013). In mei 1959 werd het Committee on Data Systems Languages, bekend als CODASYL, opgericht en gefinancierd door het Ministerie van Defensie om een nieuwe programmeertaal te ontwerpen en te ontwikkelen die voldeed aan de criteria van Mary Haws. Deze taal kreeg de naam COmmon Business-Oriented Language, of COBOL (Abby, 2023). De ontwikkeling van COBOL was gebaseerd op drie pijlers: leesbaarheid (de code moet leesbaar zijn voor zowel programmeurs als leken), overdraagbaarheid (toepassingen moeten snel en gemakkelijk kunnen worden overgezet naar een ander systeem) en flexibiliteit (de taal moet modulair genoeg zijn om zich aan te passen aan veranderende behoeften en technologieën) (Abby, 2023). In december 1960 slaagden zij erin COBOL te draaien op zowel een

UNIVAC II systeem als een RCA machine (National Museum of American History, 2013).

2.1.2. De huidige staat van COBOL

Momenteel wordt COBOL nog steeds gebruikt. De studie van Reuters (2017) toont aan dat 43 procent van de bankindustrie nog steeds steunt op COBOL. Sterker nog, 95 procent van de pinautomaten steunen op de programmeertaal. Volgens de studie van Micro Focus (2022) staan er momenteel 800 miljard lijnen COBOL in productie. Daarnaast geven de bevraagden ook aan dat de hoeveelheid COBOL in productie nog zou verhogen. Voor 92 procent van de bevraagden blijven de COBOL-applicaties een strategische asset, die met der tijd ook gemoderniseerd zullen worden. Een voorbeeld hiervan is het integreren van cloud omgevingen en COBOL-applicaties (Micro Focus, 2022).

2.2. Native VS cross-platform development

2.2.1. Wat is native development?

Er wordt van native development gesproken wanneer een programmeur een toepassing ontwikkelt die alleen op een bepaald platform kan draaien, zoals iOS of Android. Hiervoor moet de programmeur de programmeertaal en tools (Marchuk, g.d.) gebruiken die het gekozen platform biedt. Om native Android-toepassingen te schrijven, moet de ontwikkelaar Java of Kotlin gebruiken. Talen die native door het iOS-platform worden ondersteund zijn Objective-C en Swift (Schmitt, 2022).

2.2.2. Wat is cross-platform development

Cross-platform ontwikkeling, ook bekend als multi-platform ontwikkeling, is een manier van ontwikkelen waarbij met één programmeertaal, en dus één codebase, een applicatie kan worden ontwikkeld die op verschillende platforms kan draaien (Kotlin Foundation, 2022).

2.2.3. Een vergelijking tussen native en cross-platform development

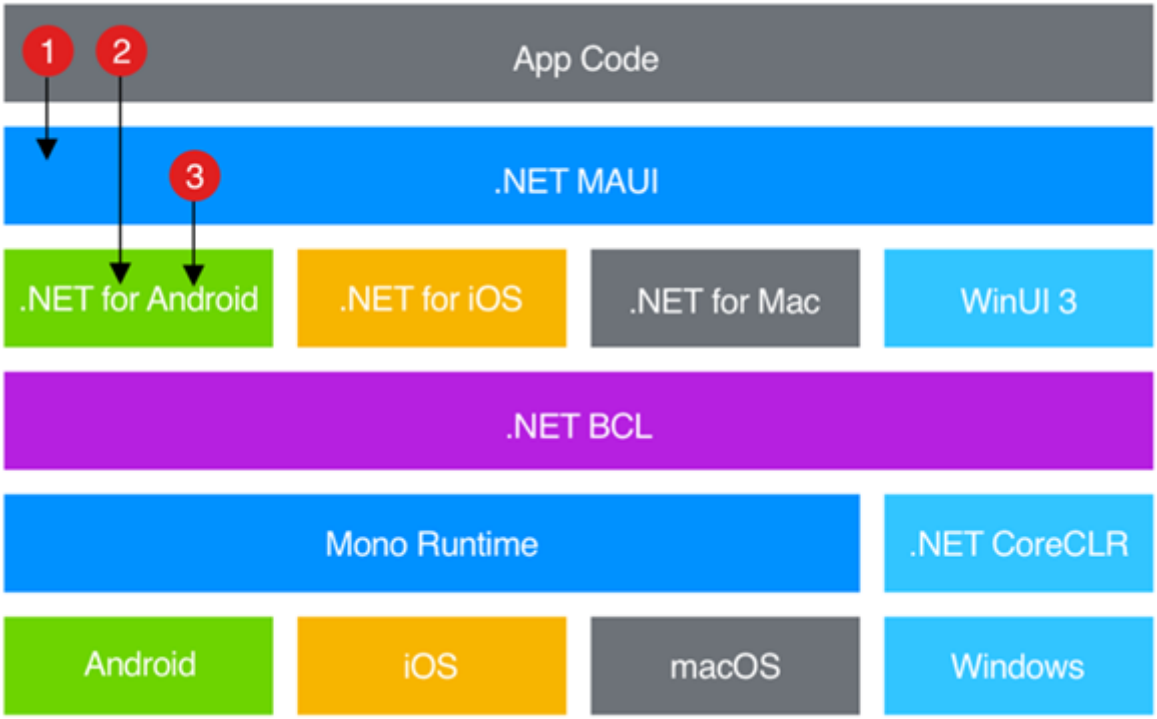
Voordelen	Nadelen
Hoge performantie De programmeertaal en API's van het doelplatform zijn geoptimaliseerd voor maximale prestaties.	Hogere ontwikkelkost Als een applicatie voor zowel iOS als Android moet worden ontwikkeld, moeten er twee verschillende teams van ontwikkelaars aan werken.
Uniforme UX (User eXperience) Elke native applicatie gebruikt dezelfde interface-elementen die door het platform worden geleverd. Hierdoor zien apps er consistenter uit en gedragen ze zich consistenter. Dit maakt een nieuwe applicatie intuïtiever en sneller te gebruiken.	Grotere ontwikkelteams Zoals gezegd zijn er meerdere teams nodig om verschillende applicaties voor de verschillende platforms te ontwikkelen. Binnen deze teams zal een groot aantal experts nodig zijn om hun expertise in te brengen tijdens het ontwikkelingsproces.
Toegang tot de gehele featureset van een platform Zoals vermeld in het eerste voordeel, kan native ontwikkeling gebruik maken van de API's die het platform biedt. Dit kan het gebruik zijn van de camera, GPS, pushmeldingen etc.	Verschillende codebases met risico op meer bugs Aangezien een applicatie meerdere programmeertalen vereist, is het gevolg dat er verschillende codebases zullen zijn. Dit kan leiden tot meer bugs omdat er meer regels code nodig zijn.
	Native applicaties kunnen verschillende logica-implementaties hebben Omdat elk platform een andere native programmeertaal biedt, kan het implementeren van bepaalde logica tot fouten leiden omdat de taal de logica anders interpreteert. Hetzelfde artikel kan bijvoorbeeld een verschillende prijs hebben op de twee platforms omdat de prijsberekening anders is geïmplementeerd.

Tabel 2.1: De voor- en nadelen van native development (Kotlin Foundation, 2023)

Voordelen	Nadelen
Deelbare code Met cross-platform ontwikkeling kunnen ontwikkelaars meerdere platforms bereiken met één enkele codebasis. Dit maakt een stuk code ook overdraagbaar van het ene project naar het andere.	Lagere performantie Omdat de cross-platform taal meerdere platformen moet kunnen aanspreken, kan de taal niet volledig worden geoptimaliseerd voor een bepaald platform, waardoor de prestaties op één platform iets langzamer zijn.
Sneller ontwikkelproces Er hoeven minder regels code te worden geschreven en getest, wat het ontwikkelingsproces aanzienlijk versnelt.	Beperkte toegang tot de gehele featureset van een platform Sommige platformen geven de moedertaal alleen toegang tot bepaalde API's of functies. Een voorbeeld hiervan is het weigeren van toegang tot pushmeldingen.
Hoge kosteneffectiviteit Kleinere ontwikkelingsteams kunnen meerdere platforms met dezelfde programmeertaal aanspreken, waardoor de ontwikkelingskosten dalen.	Beperkte UX-consistentie Cross-platform programmeertalen bieden elk hun eigen interfacecomponenten. Dit betekent dat cross-platform toepassingen vaak verschillende UI's hebben.
Gedeelde logica Het feit dat de toepassing is ontwikkeld met een enkele codebase betekent dat kan worden aangenomen dat de geïmplementeerde logica consistent is voor alle platforms.	

Tabel 2.2: De voor- en nadelen van cross-platform development (Kotlin Foundation, 2023)

Zoals uit Tabel 2.1 Kotlin Foundation (2023) en Tabel 2.2 Kotlin Foundation (2023) op te maken is, is het debat over native VS cross-platform ontwikkeling niet eenvoudig. Elk heeft zijn voor- en nadelen, en voor elk project moet een grondige analyse worden gemaakt om te bepalen welke van deze zaken het belangrijkst zijn voor het project. Als een project hoge prestaties vereist en hoge ontwikkelingskosten minder belangrijk zijn, kan ervoor gekozen worden om de toepassing in de native programmeertaal te ontwikkelen.
Als het project daarentegen door een klein team wordt ontwikkeld en de applicatie zo snel mogelijk op beide platforms beschikbaar moet zijn, kan voor cross-platform ontwikkeling worden gekozen.



Figuur (2.1)
Schematische voorstelling van de bouwstenen van .NET MAUI (Britch, 2023)

2.3. .NET MAUI

2.3.1. Overzicht van wat .NET MAUI is en het doel ervan

.NET MAUI staat voor .NET Multi-platform App UI en is de opvolger van Xamarin.Forms. Terwijl Xamarin.Forms dient als cross-platform programmeertaal die alleen op mobiele platformen kan worden gebruikt, kan .NET MAUI ook worden gebruikt om desktop en Smart TV applicaties te ontwikkelen. NET MAUI is vanaf de grond opgebouwd met uitbreidbaarheid en prestaties in het achterhoofd. Xamarin en .NET MAUI hebben veel gelijkenissen, maar de laatste maakt het ook mogelijk om platformspecifieke code toe te voegen (Brady Gaster, 2020).

2.3.2. De werking van .NET MAUI

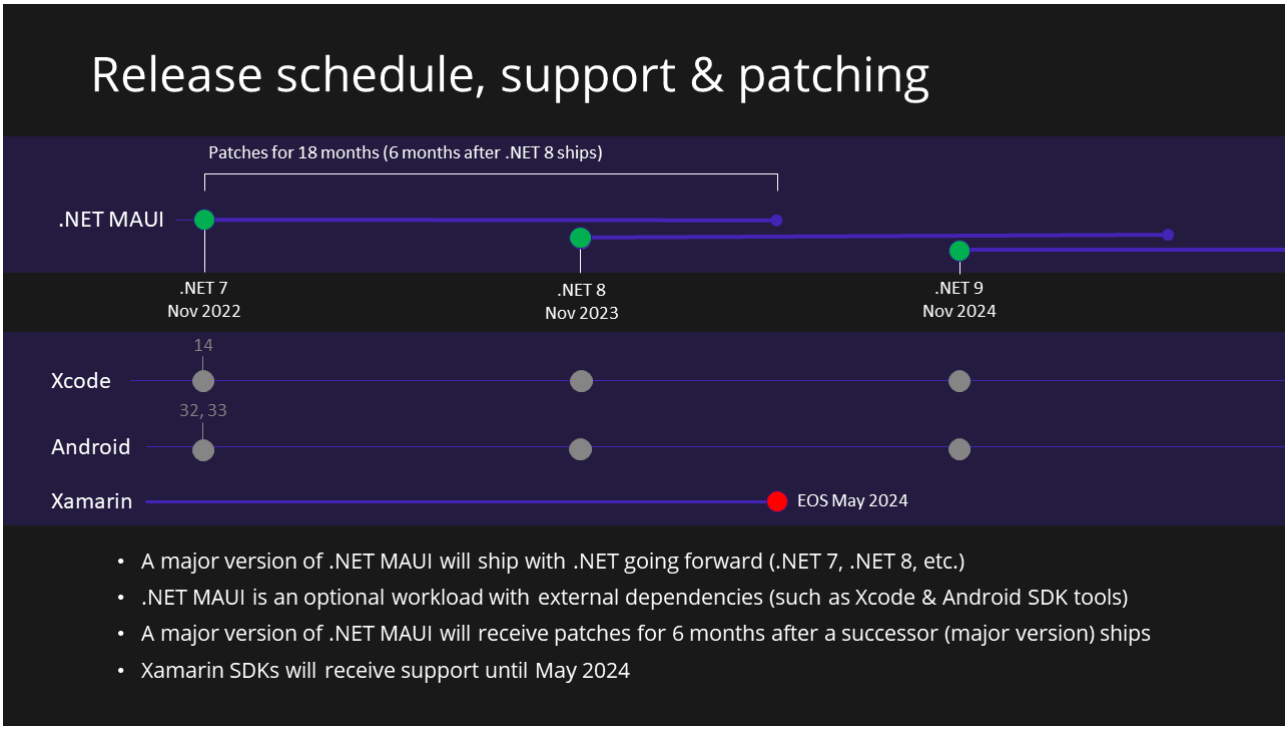
.NET MAUI voorziet een framework per platform waarmee in C# geschreven code kan worden gecompileerd voor het gewenste platform. Momenteel heeft .NET MAUI vier platformspecifieke frameworks: .NET voor Android, .NET voor iOS, .NET voor Mac en de Windows UI 3 bibliotheek. Elk van deze frameworks gebruikt de .NET Base Class Library, kortweg BCL, die fungeert als abstractie tussen de geschreven broncode en de verschillende platformen. De BCL vertrouwt op de .NET runtime om de toepassing te compileren. Ook hier zijn verschillende platformspecifieke tools voorzien. Zo is er de Mono runtime voor Android, iOS en macOS, en de

.NET CoreCLR voor Windows desktop applicaties (Britch, 2023).
Met BCL kan een toepassing worden gemaakt die op verschillende platforms draait met dezelfde codebasis. Elk platform heeft echter zijn eigen manier om de gebruikersinterface te definiëren, evenals de manier waarop de verschillende elementen met elkaar communiceren en interageren. Dit is waar .NET MAUI u de keuze geeft om verschillende gebruikersinterfaces voor verschillende platformen te creëren of, omgekeerd, één gebruikersinterface voor alle platformen te creëren (Britch, 2023).

2.3.3. Een vergelijking tussen .NET MAUI en Xamarin

Aangezien .NET MAUI de opvolger is van Xamarin.Forms, zijn er veel overeenkomsten, maar ook veel verschillen. In deze sectie worden de twee frameworks met elkaar vergeleken.
Het eerste grote verschil tussen .NET MAUI en Xamarin is de projectstructuur. In Xamarin wordt bij het aanmaken van een project een projectmap per platform aangemaakt, met als gevolg dat platformspecifieke code en resources (zoals fonts, afbeeldingen...) per project moeten worden onderhouden. Met de .NET MAUI wordt een projectmap aangemaakt die meerdere platformmappen bevat. In deze mappen wordt platformspecifieke code opgeslagen (Koleva, 2023).
Het volgende verschil tussen de twee frameworks is de build tooling, .NET MAUI kan de cross-platform .NET CLI (Command Line Interface) gebruiken om projecten te compileren en uit te voeren. Terwijl Xamarin.Forms het .NET framework moet gebruiken om applicaties te bouwen (Kathiresan, 2022).
Een derde belangrijk verschil tussen Xamarin.Forms en de .NET MAUI zijn de platformspecifieke API's. Xamarin biedt een enkele cross-platform API die werkt op Android, iOS en Windows. Met .NET MAUI wordt voor elk platform een API geleverd, waardoor ontwikkelaars kunnen profiteren van de functies van elk platform (UXDivers, g.d.).
Ten slotte kan er met .NET MAUI gekozen worden om de GUI (Graphical User Interface) te bouwen met HTML (HyperText Markup Language) of XAML (Extensible Application Markup Language). Terwijl je met Xamarin.Forms alleen met XAML kan werken (Kathiresan, 2022).

2.3.4. Een blik op de toekomst van .NET MAUI



Figuur (2.2)
.NET MAUI release schema (Ramel, 2022)

Elke nieuwe versie van .NET zal vanaf nu ook een major versie van .NET MAUI bevatten. Elk van deze major versies zullen 18 maanden ondersteuning krijgen (Ramel, 2022).
Daarnaast zal de support voor Xamarin.Forms stopgezet worden in mei 2024. Concreet wil dit zeggen dat er geen updates of patches meer zullen uitgebracht worden voor Xamarin.Forms (Ramel, 2022).

2.4. Real-time applicaties

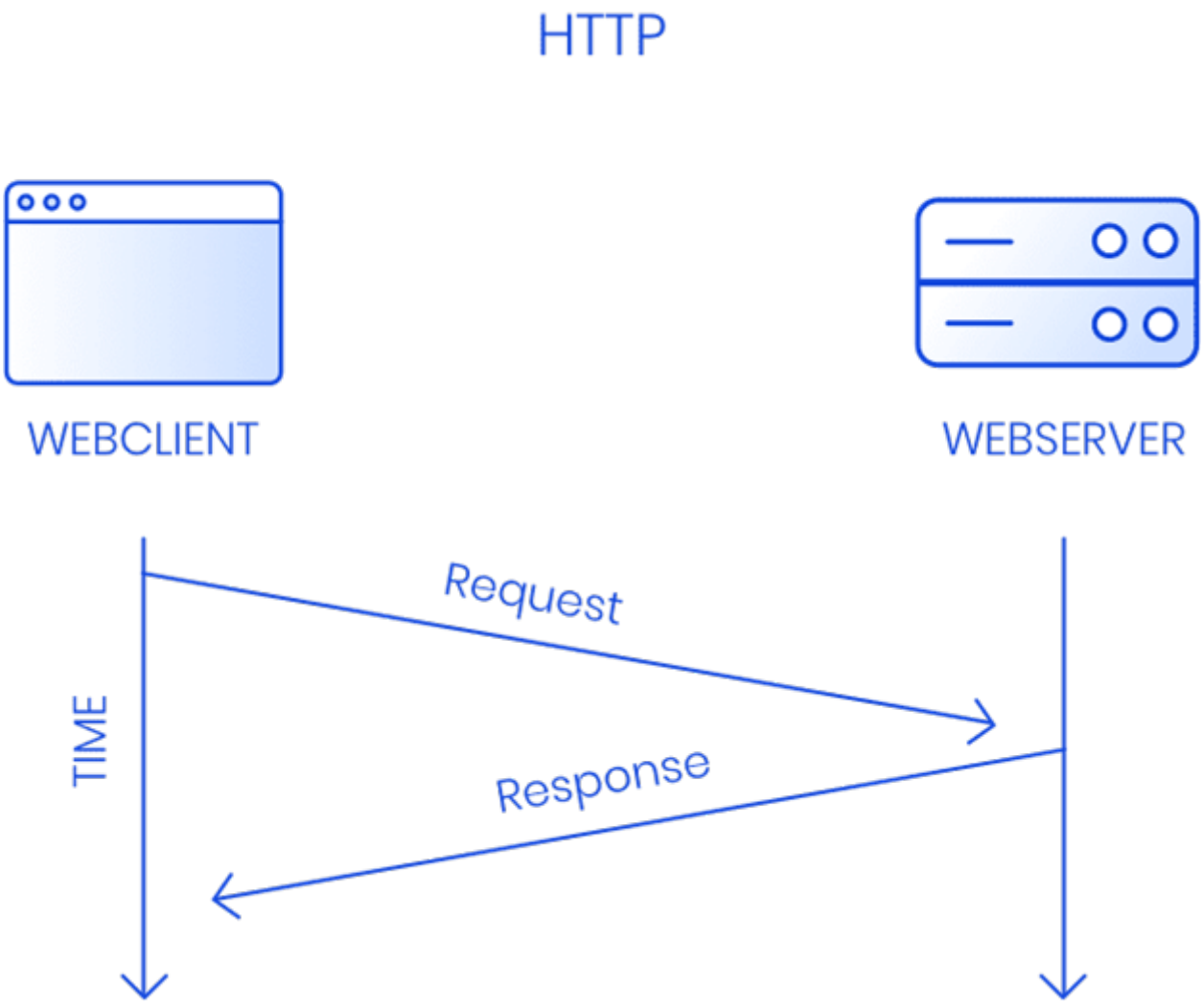
2.4.1. Definitie van een real-time applicatie

Real-time computing verwijst naar computersystemen die zijn ontworpen om gegevensinvoer in real-time te verwerken en erop te reageren, doorgaans binnen milliseconden of microseconden nadat gebeurtenissen zich hebben voorgedaan (Suse, g.d.).
De reactietijden moeten onmiddellijk lijken en het systeem moet zonder vertraging of onderbreking doorgaan (Suse, g.d.).
Real-time computing is van cruciaal belang voor toepassingen die onmiddellijke en nauwkeurige gegevensverwerking vereisen, zoals industriële controle, medische toepassingen, luchtvaart, defensie, multimedia, games en financiële handelssystemen (Suse, g.d.).

Real-time computing vereist gespecialiseerde hardware- en softwarecomponenten, waaronder real-time besturingssystemen, real-time netwerken en synchrone programmeertalen, om het systeem binnen het gewenste tijdsbestek te laten functioneren (Suse, g.d.).

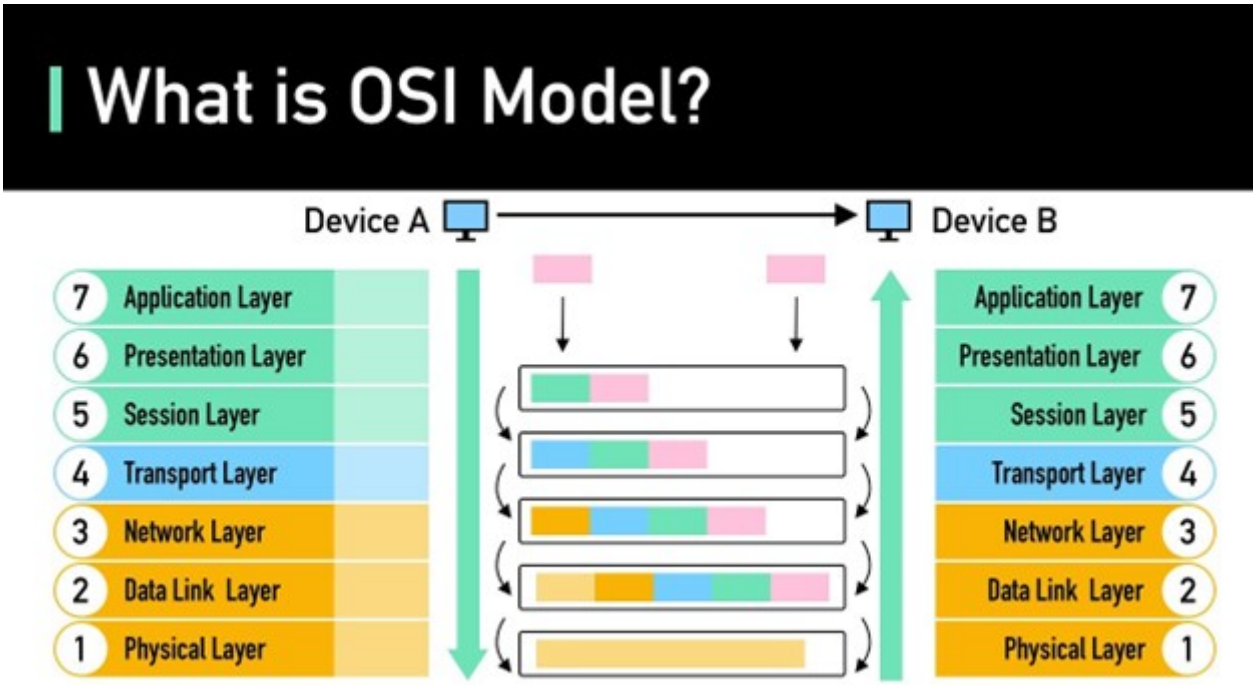
2.4.2. HyperText Transfer Protocol

HTTP is het protocol dat ten grondslag ligt aan het internet. Het protocol wordt gebruikt om gegevensbronnen (zoals HTML, XML, JSON-documenten) op te vragen bij een server (Cloudflare, g.d.).



Figuur (2.3)
Schematische voorstelling van het client-server model (Combell, g.d.)

HTTP is een client-serverprotocol, wat betekent dat verzoeken om gegevens altijd door de client (ook bekend als de ontvanger) worden geïnitieerd. Het verzoek wordt naar de server gestuurd, die op zijn beurt het gevraagde document samenstelt en terugstuurt als antwoord op het verzoek (MDN, 2023a)



Figuur (2.4)
Het OSI model (ByteByteGo, 2022)

Dit protocol is oorspronkelijk voorgesteld door Tim Berners-Lee in 1989 en is gebouwd bovenop de TCP- en IP-protocollen (MDN, 2023b).
Het HTTP-protocol bevindt zich in de zevende laag (ook toepassingslaag genoemd) van het OSI-model (MDN, 2023a)
De filosofie van HTTP rust op vier pijlers:

- Uitbreidbaarheid

  De invoering van HTTP-headers in HTTP versie 1 (HTTP/1) (MDN, 2023b) maakt het mogelijk extra informatie toe te voegen aan verzoeken en antwoorden, zolang client en server deze overeenkomst aanvaarden (Paessler, g.d.).

- Staatloos

  Staatloos betekent dat tijdens en na het verzenden en ontvangen van een verzoek en antwoord geen informatie door de server wordt bijgehouden. Met andere woorden, elk verzoek dat de server ontvangt wordt onafhankelijk gezien en uitgevoerd. Een stateless protocol is ontworpen met het oog op schaalbaarheid. Afhankelijk van de werklast kunnen dus meerdere servers worden opgezet met dezelfde informatie, en elke server in de configuratie kan het verzoek ontvangen en verwerken (Paessler, g.d.).

  Om een toestand bij te houden kunnen HTTP-cookies worden gebruikt. Dit is mogelijk dankzij de bovengenoemde uitbreidbaarheid en maakt het moge-

lijk bepaalde informatie (zoals authenticatietokens) te delen tussen meerdere verzoeken (MDN, 2023a).

- Verbindingsloos

  Het HTTP-protocol wordt om twee redenen als verbindingsloos beschouwd. Omdat verbindingen worden gemaakt door de transportlaag (laag 4 van het OSI-model), valt het niet onder de verantwoordelijkheid van een applicatie laag (laag 7 van het OSI-model) applicatie (MDN, 2023a).

  Bovendien worden voor elk verzoek verbindingen gelegd tussen de client en de server. Zodra het verzoek is verwerkt, sluit de server de verbinding (Paessler, g.d.).

- Media onafhankelijk

  Zolang de client en de server weten hoe zij bepaalde door het MIME-type (Multipurpose Internet Mail Extensions) gespecificeerde gegevens moeten verwerken, kunnen de gegevens via het HTTP-protocol worden getransporteerd (Paessler, g.d.).

De evolutie van het HTTP-protocol

HTTP/0.9

Aanvankelijk had het HTTP-protocol geen versienummer (aangegeven door het cijfer achter het teken /). Om de verschillende versies van elkaar te scheiden, kreeg de allereerste implementatie van het HTTP-protocol echter versienummer 0.9 (MDN, 2023b).

Met verzoeken die bestaan uit een enkele regel bestaande uit het type verzoek en de naam van de gegevensbron, is deze versie van HTTP vrij primitief. Ook kan met deze versie alleen een GET-verzoek worden gedaan. Bijgevolg kunnen alleen gegevens worden gelezen (Grigorik, g.d.)

HTTP/1.0 HTTP-versie 1.0 werd uitgebracht in 1990, vijf jaar nadat versie 0.9 was geïntroduceerd, en voegde nieuwe functies toe om eerdere tekortkomingen te verhelpen:

- HTTP-header

  Zoals hierboven vermeld, bestond een verzoek van versie 0.9 alleen uit de methode en de naam van de gegevensbron die werd opgevraagd (Fulber-Garcia, 2022). Door de toevoeging van de HTTP-header kunnen nu metadata worden opgenomen in verzoeken en antwoorden, wat deze versie van het protocol flexibel en uitbreidbaar maakt (MDN, 2023b).

- Versiebeheer

  Elk HTTP-verzoek bevat nu de gebruikte HTTP-versie (MDN, 2023b).

- Statuscode

  Aan elk antwoord wordt een statuscode toegevoegd. Dit is een manier voor de client om bij elk antwoord te controleren of het verzoek al dan niet correct is verwerkt (Fulber-Garcia, 2022).

- Content-type

  Zoals hierboven vermeld, kunnen met de HTTP-header extra metadata aan een verzoek worden toegevoegd. Voor HTTP/1.0 is nu een inhoudstype in de header opgenomen, waardoor ook andere documenten dan HTML kunnen worden verzonden (Fulber-Garcia, 2022).

- Extra methoden

  HTTP/1.0 heeft ten slotte twee nieuwe methoden toegevoegd, POST en HEAD. Dit zorgt ervoor dat gegevens nu zowel geschreven als gelezen kunnen worden (Fulber-Garcia, 2022).

HTTP/1.1

HTTP/1.1 werd een jaar na de release van HTTP/1.0 geïmplementeerd. Dit was slechts bedoeld als een verbetering van de 1.0-versie:

- Host header

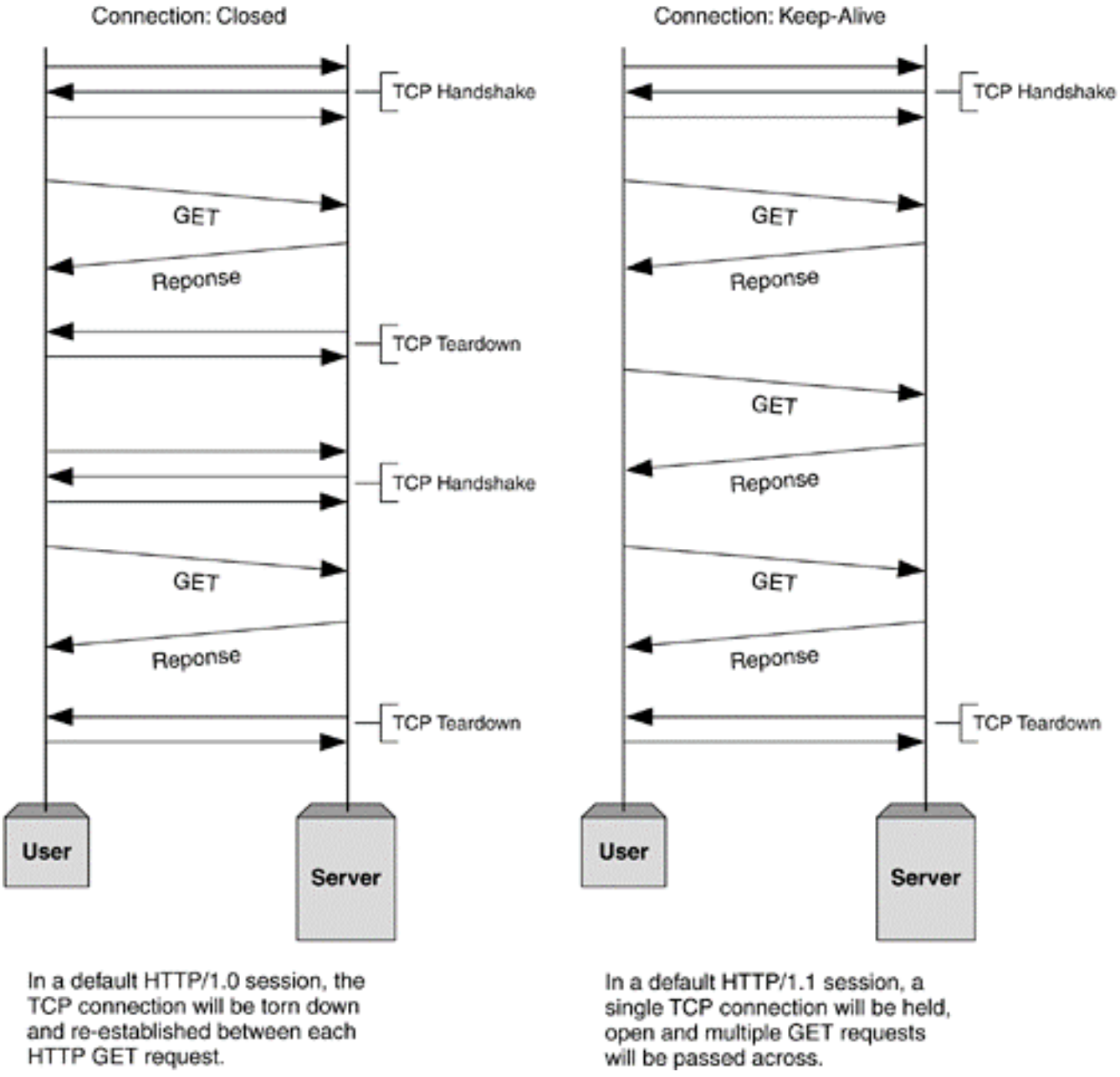
  Met HTTP/1.1 moet de client het host veld toevoegen aan de header van elke aanvraag. Vóór deze versie was dit niet verplicht. Dankzij dit veld kunnen verzoeken via een proxyserver naar de juiste server worden geleid. In de praktijk betekent dit dat als er verschillende domeinen naar dezelfde locatie verwijzen, er toch een onderscheid kan worden gemaakt om ervoor te zorgen dat het verzoek niet naar de verkeerde server gaat (Fulber-Garcia, 2022).

- Persistente verbinding

  Zoals hierboven vermeld, moest de client in HTTP-versie 1.0 een verbinding maken voor elk verzoek dat hij wilde verzenden. In versie 1.1 kan de verbinding open worden gehouden totdat de client aangeeft dat hij de verbinding wil sluiten (MDN, 2023b).

- Continue-status

  Vóór de invoering van de continue status konden sommige verzoeken niet door de server worden verwerkt. Als dit het geval was, werd het verzoek gewoon onbeantwoord afgewezen. Vanaf versie 1.1 kan een client eerst de HTTP-headers van het verzoek aan de server doorgeven. Als de server antwoordt met de status continue (statuscode 100), weet de client dat hij het hele verzoek kan doorsturen en een antwoord van de server kan verwachten (Fulber-Garcia, 2022).



Figuur (2.5)
Schematische voorstelling van het verschil tussen niet-herbruikbare en herbruikbare connecties van respectievelijk HTTP/1.0 en HTTP/1.1 (Hesham, 2019)

- Aanvullende methoden

  Versie 1.1 introduceert ook een aantal nieuwe methoden. Deze omvatten PUT, PATCH, DELETE, CONNECT, TRACE en OPTIONS (Fulber-Garcia, 2022).

HTTP/2.0
HTTP/2.0 is uitgebracht in 2015, 18 jaar na 1.1. Het is echter nog niet op grote schaal ingevoerd. Het gebruik van HTTP/2.0 is dan ook een bewuste keuze van de ontwikkelaars (Ab, g.d.). HTTP/2.0 brengt een aantal nieuwe functies die nieuwe mogelijkheden bieden, namelijk:

- Multiplexing

  In versie 1.1 van het HTTP-protocol werden verzoeken om gegevens sequentieel verwerkt. Dat wil zeggen, het eerste HTML-document werd opgevraagd en geladen, dan het tweede document, enzoverder tot alles geladen was. Dit kon echter tot problemen leiden als een bepaalde bron niet kon worden geladen omdat de rest van de bron werd opgehouden. Met HTTP/2.0 kan één enkele TCP-verbinding worden gebruikt om meerdere verzoeken tegelijk te doen. Dit zorgt ervoor dat wanneer het bovengenoemde probleem zich voordoet, de resterende gegevens al kunnen worden geladen (Cloudflare, g.d.).

- Server push

  Vóór de invoering van HTTP/2.0 kon de server alleen gegevens naar de client pushen nadat deze een verzoek had verzonden. In versie 2.0 is echter server push geïmplementeerd. Hierdoor kan de server gegevens naar de client sturen nog voordat de client een verzoek naar hem heeft gestuurd. Dit lost prestatieproblemen met lange polling op (Grigorik, 2016).

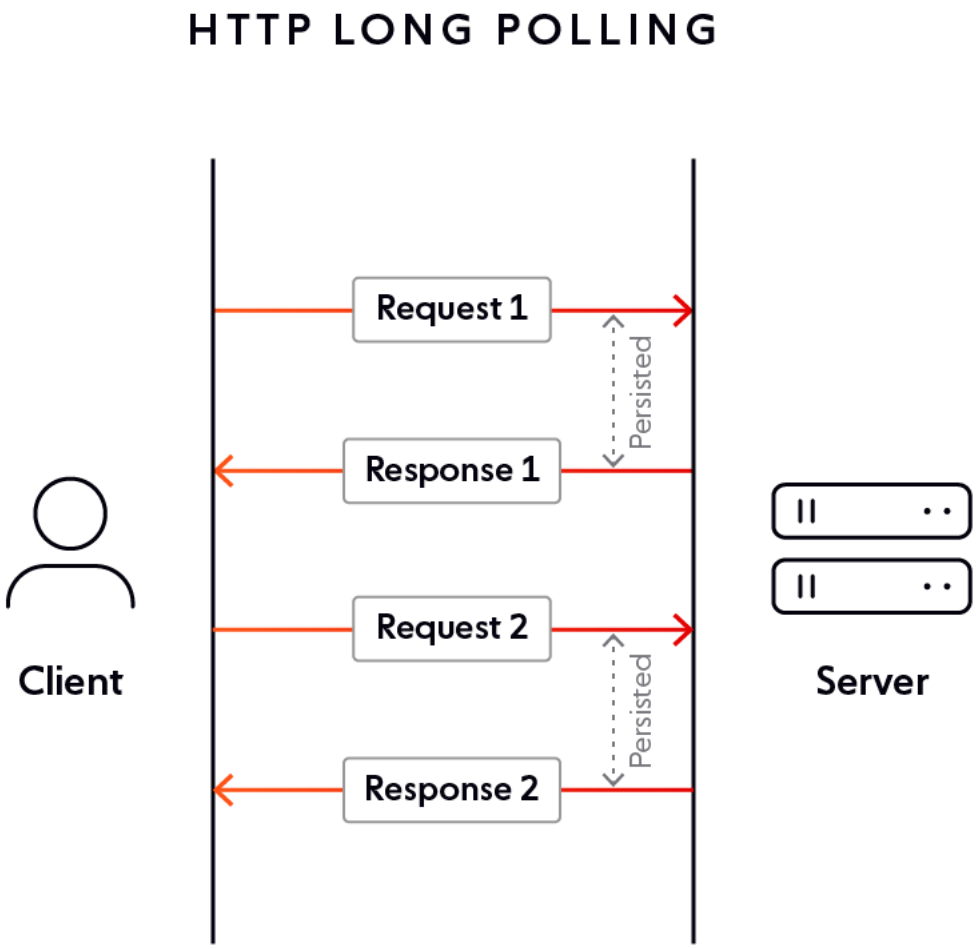
- Compressie van headers

  De toevoeging van HTTP-headers aan een verzoek vergroot de omvang van het verzoek. Dit kan leiden tot het langzamer laden van gegevens en het langzamer verwerken van gegevens door de client. Deze verzoeken kunnen kleiner worden gemaakt en sneller worden verwerkt door de headers te comprimeren. HTTP/1.1 deed dit al, maar versie 2.0 heeft een veel verfijnder compressiealgoritme, het HPACK-algoritme. Dit algoritme verwijdert alle overbodige informatie, waardoor verzoeken kleiner worden (Cloudflare, g.d.).

2.4.3. WebSocket protocol

Het WebSocket protocol is een relatief nieuw protocol. Het protocol werd voor het eerst beschreven in 2008 door Micahel Carter en Ian Hickson en kreeg algemene ondersteuning rond 2010 (Ably, 2020).
In december 2011 publiceerde het IETF (Internet Engineering Task Force) de whitepaper die het WebSocket protocol beschreef onder de titel “RFC 6455 - The WebSocket Protocol” (Fette & Melnikov, 2011).

Volgens de paper van Fette en Melnikov (2011) is het WebSocket protocol ontwikkeld om real-time applicaties mogelijk te maken zonder het zogenaamde “HTTP long polling” te misbruiken.
Bij HTTP long polling stuurt de client een aanvraag uit naar de server. In plaats van direct te antwoorden en daarna de connectie te sluiten, wacht de server met antwoorden totdat er nieuwe data beschikbaar is. Na het uitsturen van het antwoord, krijgt de server meteen een nieuwe aanvraag van de client (Singh, g.d.).
Het WebSocket protocol is full-duplex en laat bidirectionele communicatie toe. Concreet betekent dit dat in plaats van een aanvraag te sturen die enkel beantwoord wordt wanneer er nieuwe data is, het nu mogelijk is om data door te sturen naar de client zonder dat deze een aanvraag verstuurd heeft. Dit zorgt voor een betere performantie ten opzichte van het HTTP long polling (Fette & Melnikov, 2011)



Figuur (2.6)
Schematische voorstelling van HTTP long polling (Ably, 2020)

2.4.4. Real-time programmeren binnen het .NET ecosysteem

De ASP.NET SignalR bibliotheek is beschikbaar voor real-time applicatieontwikkeling binnen het .NET ecosysteem. Deze bibliotheek maakt het mogelijk een tweerichtingscommunicatie op te zetten tussen een client en een server. De client kan met name bepaalde functies op de server aanroepen, maar de server kan ook functies opvragen bij de client (Brady Gaster, 2020).

In het traditionele client-server model kan alleen de client functies op de server aanroepen. Dit betekent dat als de gegevens zo actueel mogelijk moeten zijn, er extra logica moet worden geïmplementeerd om de server na een bepaald aantal seconden te polsen naar de (mogelijk) gewijzigde gegevens (deze implementatie wordt ook wel "long polling"genoemd). Dit kan leiden tot slecht presterende toepassingen (Brady Gaster, 2020).

Dankzij het gebruik van SignalR is het nu ook mogelijk dat de server functies gaat aanroepen op de client. Dit zorgt ervoor dat het hierboven getoonde voorbeeld op een veel performantere manier kan worden geïmplementeerd. Wanneer gegevens op de server worden gewijzigd, kan de server alle luisterende clients opdragen de nieuwe gegevens op te halen. Dit zorgt ervoor dat gegevens alleen worden opgevraagd wanneer de gegevens veranderen, in plaats van telkens na een bepaalde periode (Brady Gaster, 2020).

3

Methodologie

Inleiding

Tijdens het opstellen van deze bachelorproef hebben twee grote activiteiten plaatsgevonden, namelijk een literaire studie en het opstellen van een proof-of-concept.

- Literaire studie

  De literaire studie heeft als doel een gepast referrentiekader te geven voor elke lezer van deze proef.

- proof-of-concept

  Het doel van de proof-of-concept is om een conclusie te kunnen vormen op de onderzoeksvraag.

In dit hoofdstuk wordt de wijze waarop deze twee activiteiten zijn uitgevoerd verduidelijkt.

3.1. Theoretisch onderzoek

3.1.1. Fase 1

Het doel van de eerste fase van het literaire onderzoek was het bepalen van de in dit onderzoek op te nemen onderwerpen.
Hierbij was het van groot belang dat de lezer van deze proef de juiste contextualisering krijgt. Daarnaast was het ook belangrijk om enkel de juiste onderwerpen te vermelden.

3.1.2. Fase 2

De doelstelling van deze fase was onderzoek te voeren naar de geschiedenis van de COBOL programmeertaal en hoeveel er nog gebruik gemaakt wordt van deze programmeertaal in de huidige technologische markt.

3.1.3. Fase 3

Tijdens deze fase werd er onderzocht wat native en cross-platform development is en welke voor- en nadelen deze twee manieren van werken met zich meebrengen.

3.1.4. Fase 4

Tijdens fase 4 werd er onderzoek gedaan naar wat .NET MAUI is, hoe het werkt, hoe het zich verhoudt ten op zichte van zijn voorganger Xamarin.Forms en hoe de toekomst eruitziet voor het nieuwste framework van Microsoft.

3.1.5. Fase 5

Bij het uitvoeren van het onderzoek voor deze fase werd er getracht om eerst de term “real-time applicatie” te definiëren, waarna werd onderzocht wat het HTTP protocol is en hoe dit geëvolueerd is ten opzichte van de eerste versie. Daarna werd onderzocht wat het WebSocket protocol is en waarom dit ontwikkeld is. Tot slot werd de SignalR bibliotheek van Microsoft bestudeerd in verband met het real-time programmeren binnen .NET.

3.2. Proof-of-concept

Na het uitvoeren van het literaire onderzoek werd de proof-of-concept opgezet. Ook dit werd gefaseerd uitgevoerd.

3.2.1. Fase 1

Tijdens deze fase werd eerst en vooral een opsomming gemaakt van welke applicaties er nodig zijn om de proof-of-concept volledig op te kunnen zetten en welke apparatuur en software hiervoor noodzakelijk zijn.
Daarna werd de systeemarchitectuur opgesteld. Hiervoor werd een schema gemaakt van welke apparaten met elkaar in verbinding zouden staan en welke applicaties op welke apparaten zouden draaien.
Tot slot werd het domeinklassediagram ontworpen. Dit diagram toont welke klassen er nodig zijn, welke associaties, eigenschappen en methoden deze hebben.

3.2.2. Fase 2

In fase 2 werd het domeinklassediagram omgezet in concrete klassen en werden enkele van deze klassen getest om te controleren of de geïmplementeerde logica het verwachte resultaat oplevert.

3.2.3. Fase 3

Deze fase had als doelstelling het volledig implementeren van de applicaties die in de winkel moeten draaien, namelijk: de management API, de operations API en de operations client.

De management API moet een winkel-, printer- en staffelobject kunnen configureren en persisteren in de databank.
De operations API moet een verkoopobject kunnen aanmaken, het totaal te betalen bedrag kunnen berekenen, de digitale signalen, uitgestuurd door het python script, ontvangen en de tellerstanden van de printers op de GUI updaten.
De operations client dient als grafische interface voor de winkelverantwoordelijke. Deze moet toelaten om de huidige tellerstand van de printers te bekijken, het commando uitsturen om het totaal te betalen bedrag te berekenen, een verkoop te registreren en bij het afsluiten van het systeem moet er een aanvraag verstuurd worden naar de reporting API.

3.2.4. Fase 4

Tijdens deze fase werd het python script geschreven. Dit script moest dienen om te luisteren naar de seriële verbinding tussen de Arduino en Raspberry Pi en indien er boodschappen staan te wachten, deze berichten decoderen. Eens deze gedecodeerd zijn wordt er een PATCH-aanvraag verstuurd naar de operations API met als parameter het interfacenummer waarop het analoge signaal geregistreerd werd.

3.2.5. Fase 5

Bij het uitvoeren van fase 5 was het de bedoeling om de dagrapporten applicaties volledig te implementeren. Concreet betekent dit dat de reporting API werd opgesteld. Deze heeft als doel om een nieuw dagrapport voor een winkel te genereren, te persisteren in de databank en een push-notificatie te versturen naar de reporting client.
Daarnaast werd ook de reporting client ontwikkeld. Dit is een mobiele applicatie die dient als dashboard waarop de zaakvoerder bepaalde datavisualisaties kan bekijken omtrent het aantal verkopen in de copycentra. Deze moet de push-notificatie verzonden door de reporting API ontvangen en een melding geven aan de zaakvoerder dat er een nieuw dagrapport beschikbaar is.

3.2.6. Fase 6

Na het opstellen van de proof-of-concept is er ook nog een extra tool ontwikkeld om metingen uit te kunnen voeren. Deze verstuurt een digitaal signaal analoog aan het signaal dat verstuurd wordt door de Arduino. Het tijdstip waarop het signaal verstuurd en beantwoord wordt, wordt geregistreerd. Daarna wordt het tijdsverschil tussen deze twee punten berekend en weggeschreven naar een CSV-file (Comma Seperated Value). Met andere woorden, deze tool simuleert het python listener script en meet de responsetijd van de .NET MAUI applicatie.

4

Proof-of-concept

Inleiding
Dit hoofdstuk licht het ontwerp en de opbouw van de proof-of-concept toe.

4.1. Toelichting

Bij de voorbereiding van de proof-of-concept zijn bepaalde beslissingen genomen met betrekking tot het ontwerp en de architectuur van het project. Deze toelichting heeft als doel uit te leggen en te verduidelijken waarom die beslissingen zijn genomen.

- Gebruik van een Arduino

  Elk van de printers in een kopieercentrum is aangesloten op de Arduino UNO. De Arduino UNO is gekozen omdat deze volledig programmeerbaar is. Op dit moment pikt deze Arduino gewoon de analoge klik van een printer op en stuurt een digitaal signaal met een kleine payload naar de Raspberry Pi. In deze payload zit alleen het pinnummer waarop de klik is ontvangen. In de toekomst kan deze payload worden uitgebreid met meer informatie. Dit is mogelijk dankzij de programmeerbaarheid van de microcontroller.

- Gebruik van een Raspberry Pi

  Het gebruik van een Raspberry Pi als server heeft een aantal voordelen ten opzichte van een standaardcomputer. Eerst en vooral is de Raspberry Pi voordeliger in gebruik aangezien deze een ARM-processor gebruikt. Dit komt omdat de Raspberry Pi een ARM-processor gebruikt. Deze is veel goedkoper dan een normale processor.

  Het tweede grote voordeel van de Raspberry Pi is zijn energieverbruik. In feite is zijn stroomverbruik veel lager dan dat van de gemiddelde desktopcomputer. Dit is vooral te danken aan de eerdergenoemde ARM-processor.

Tot slot is ook de flexibiliteit van een Raspberry een groot voordeel. Dankzij de flexibiliteit van de Raspberry Pi kan hij in het proof of concept voor verschillende dingen worden gebruikt zoals onder meer om te luisteren naar de digitale signalen die de Arduino UNO uitzendt en om een aantal microservice containers in Docker te draaien.

- Toepassen van de microservices architectuur

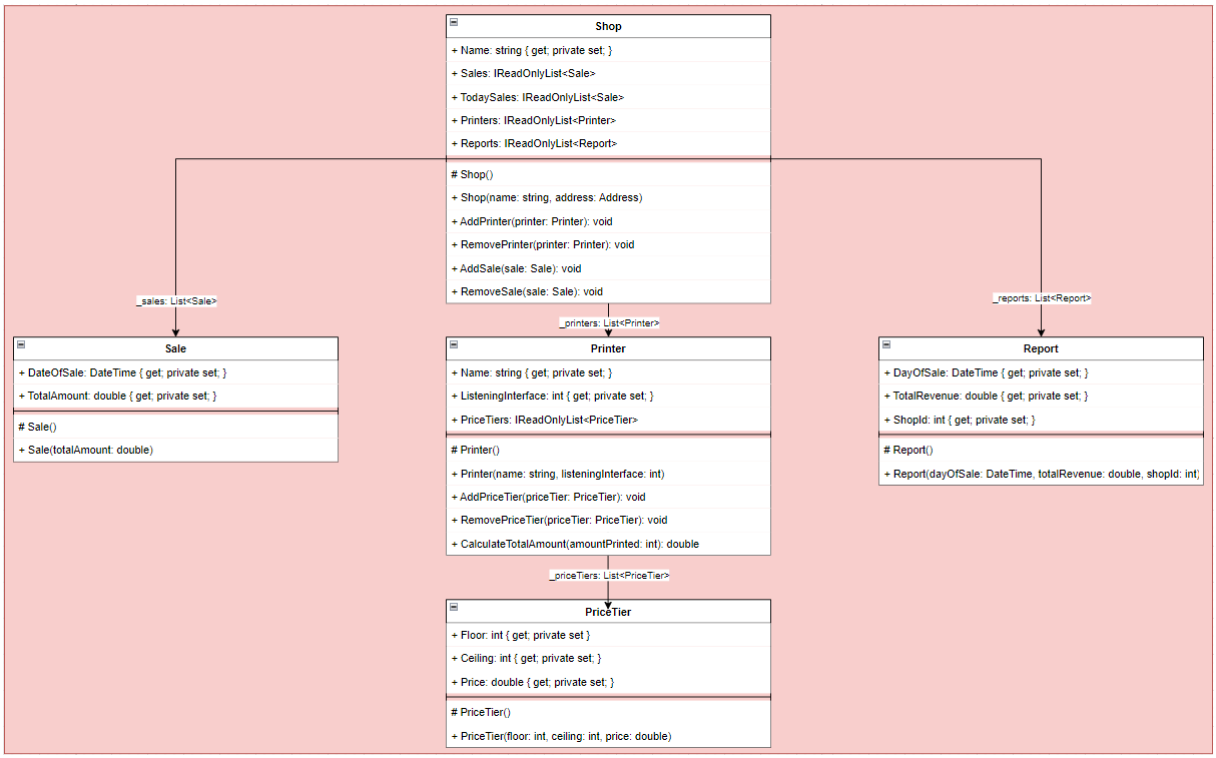
  De toepassing van de microservices architectuur brengt een paar voordelen met zich mee, namelijk: schaalbaarheid en veerkrachtigheid.

  Eerst en vooral zijn microservices heel erg schaalbaar. Goossens NV heeft plannen om de copycentra in de toekomst uit te breiden. Dit kan ervoor zorgen dat de operations API overbelast kan raken op piekmomenten. Dankzij het feit dat deze API opgezet is als microservice, is het mogelijk om deze service horizontaal te schalen. Concreet wil dit zeggen dat er meerdere instanties van deze service naast elkaar zullen draaien en worden de aanvragen over de verschillende instanties gespreid. Wanneer het piekmoment voorbij is, worden de overtollige services terug verwijderd.

  Daarnaast zijn microservices ook heel erg veerkrachtig. Tijdens de proof-of-concept werden twee verschillende API's gebouwd. Terwijl een eerste wordt gebruikt om de printers en diens staffels te managen, dient een tweede om de dagelijkse operaties te ondersteunen. In de toekomst kunnen er extra API's worden ontwikkeld en uitgerold in de winkels. Dankzij de veerkrachtigheid van de microservices kan één van de microservices crashen of offline gehaald worden zonder de andere services in het gedrang te brengen.

- Gebruik van python bij het listener script Python heeft ingebouwde bibliotheken voor het lezen van en schrijven naar de seriële bus in Python. Omdat de Arduino serieel verbonden is met de Raspberry Pi waarop de listener zal draaien, is het luisteren naar seriële bussen belangrijk. Het doel van deze luisteraar is het lezen van seriële busberichten en het decoderen van de payload in het bericht. Bibliotheken voor het versturen van verzoeken naar een REST API zijn ook opgenomen in de programmeertaal Python. Na decodering en validering van de ontvangen payload moet het script een PATCH-verzoek sturen naar de Operation API met het nummer dat uit de payload is verkregen als parameter.

4.2. Domeinarchitectuur



Figuur (4.1)
Schematische voorstelling van het domein

Het domein van deze proof-of-concept bestaat uit vijf klassen: Shop, Sale, Printer, Report en PriceTier.

- Shop

  De Shop klasse heeft volgende eigenschappen:

  - Naam (*Name*-property): de naam van de winkel.
  - Verkopen (*Sales*-property): een lijst van verkopen die plaatsgevonden hebben in de winkel.
  - Verkopen van vandaag (*TodaySales*-property): een sublijst van verkopen die vandaag hebben plaatsgevonden in de winkel.
  - Printers (*Printers*-property): een lijst van aanwezige printers in de winkel.
  - Reports (*Reports*-property): een lijst van rapporten gekoppeld aan de winkel.

  Als ook volgende methoden:

  - Toevoegen van een printer (*AddPrinter*-method): een methode voor het toevoegen van een printer aan de winkel.

  - Verwijderen van een printer (*RemovePrinter*-method): een methode voor het verwijderen van een printer uit de winkel.
  - Toevoegen van een verkoop (*AddSale*-method): een methode voor het toevoegen van een verkoop aan de winkel.
  - Verwijderen van een printer (*RemovePrinter*-method): een methode voor het verwijderen van een verkoop uit de winkel.

- Sale

  De Sale klasse bevat volgende eigenschappen:

  - Datum van verkoop (*DateOfSale*-method): de datum waarop de verkoop heeft plaatsgevonden.
  - Totaal betaald bedrag (*TotalAmount*-method): het totaal betaald bedrag.

- Printer

  De Printer klasse bestaat uit deze eigenschappen:

  - Naam (*Name*-property): de naam van de printer.
  - Geregistreerd Arduino pinnummer (*ListeningInterface*-property): het pinnummer waarop de printer geconnecteerd is met de Arduino UNO.
  - Prijsstaffels (*PriceTiers*-property): een lijst van prijsstaffels voor de printer.

  En volgende methoden:

  - Toevoegen van een prijsstaffel (*AddPriceTier*-method): een methode voor het toevoegen van een prijsstaffel aan een printer.
  - Verwijderen van een prijsstaffel (*RemovePriceTier*-method): een methode voor het verwijderen van een prijsstaffel uit een printer.
  - Bereken totaal te betalen bedrag (*CalculateTotalAmount*-method): een methode voor het berekenen van de prijs op basis van de huidige actieve prijsstaffel.

- Report

  De Report klasse heeft deze eigenschappen:

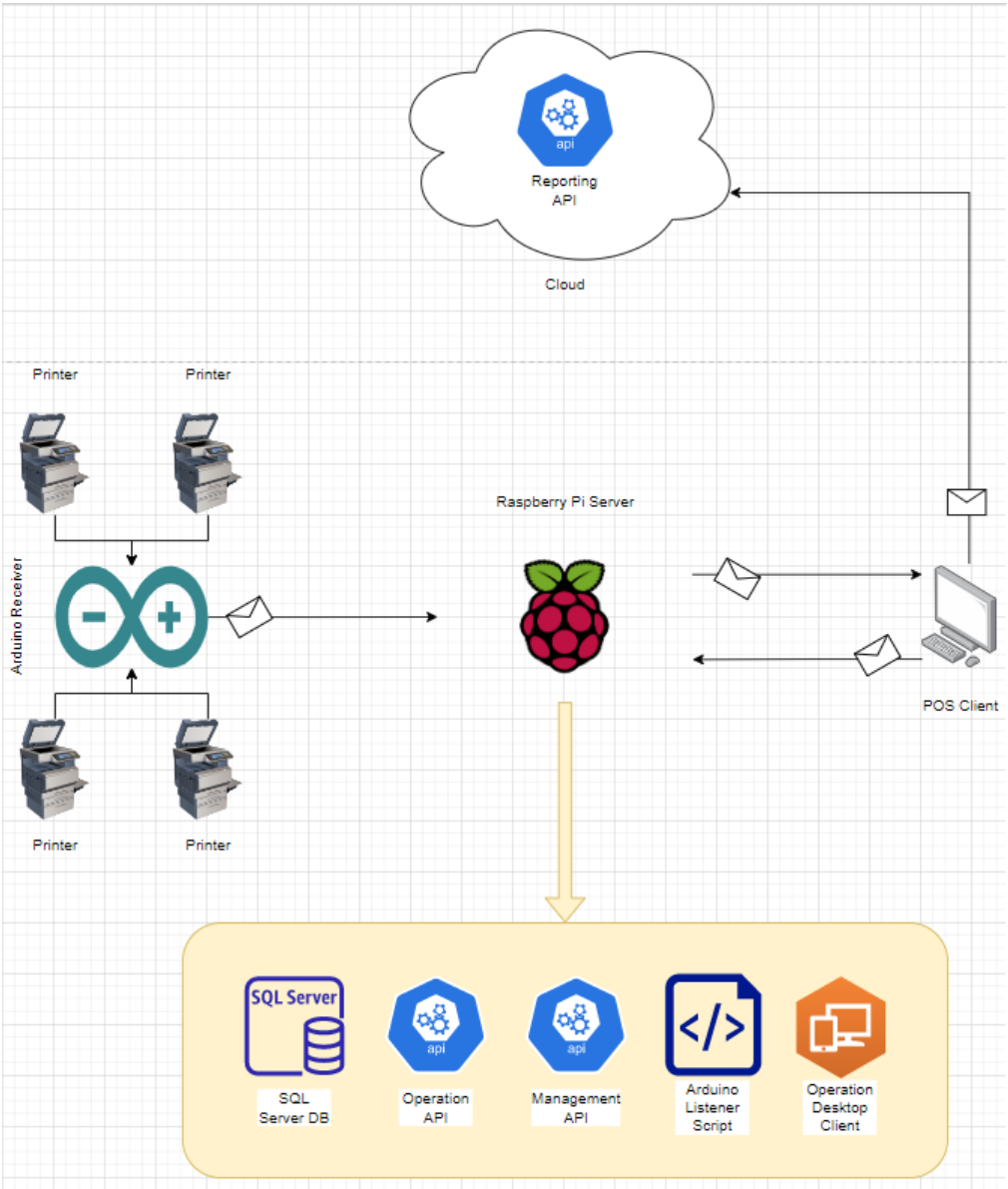
  - Datum van verkoopdag (*DayOfSale*-property): de datum van het dagrapport.
  - Totale opbrengst (*TotalRevenue*-property): de totale omzet die gedraaid werd die dag.
  - Winkel identificatie (*ShopId*-property): het unieke nummer waarmee de winkel, waarover het dagrapport gaat, identificeerbaar is.

- PriceTier

  De PriceTier klasse omvat volgende eigenschappen:

  - Ondergrens (*Floor*-property): de inclusieve ondergrens van de staffel.
  - Bovengrens (*Ceiling*-property): de inclusieve bovengrens van de staffel.
  - Prijs (*Price*-property): de prijs gekoppeld aan deze staffel.

4.3. Systeemarchictectuur



Figuur (4.2)
Schematische voorstelling van de systeemarchitectuur

Deze proof-of-concept is gebasseerd op de huidige systeemarchitectuur binnen de copycentra van Goossens NV. Momenteel zijn alle printers binnen één copycen-

ter allemaal geschakeld aan één schakelbord. Dit bord staat dan op zijn beurt in verbindingen met het kassasysteem. Het doel van dit bord is om de analoge klik die een printer uitstuurt na het printen van een bladzijde op te vangen en door te geven aan het kassasysteem.
In de nieuwe systeemarchitectuur (afgebeeld in Figuur 4.1) staan alle printers geschakeld aan een Arduino Uno. Het doel van deze Arduino is het opvangen van het analoge signaal en een digitaal signaal uit te sturen via een seriële verbinding. Deze seriële verbinding mondt uit in een Raspberry Pi, waar verschillende microservices draaien:

- Microsoft SQL Server

  De Microsoft SQL Server service fungeert als centrale databank waar alle microservices hun data kunnen wegschrijven en ophalen.

- Management API

  De Management API heeft als doel het managen van de printers en hun staffels. Onder het managen van printers wordt het aanmaken en verwijderen van printers binnen één winkel begrepen. Het managen van de printerstaffels houdt het aanmaken, verwijderen en bijwerken van printerstaffels voor één printer in.

- Arduino Listener script

  Dit script is geschreven in python en heeft als doel de seriële bus van de Raspberry Pi uit te lezen. Op deze seriële bus staan de boodschappen uitgestuurd door de Arduino Uno. Ieder bericht bevat een kleine payload die aangeeft welke op welke interface van de Arduino de analoge klik is binnengekomen. Na het decoderen van de payload, stuurt het script een PATCH-aanvraag naar de Operation API.

- Operations API

  Deze API is bedoeld om de dagdagelijkse operaties te ondersteunen. Deze haalt alle data op die nodig is voor de desktop kassa-applicatie, berekent het totaal te betalen bedrag, schrijft alle verkopen weg naar de databank, ontvangt de PATCH-aanvragen van het Arduino Listener script en stuurt update commando's uit naar de desktopapplicatie.

- Operation Desktop Client

  Dit is bedoeld als GUI (Graphical User Interface) voor de verko(o)p(st)ers van de copycentra. Hier wordt de huidige tellerstand van de printers bijgehouden. Indien een klant wil afrekenen, kan de winkelverantwoordelijke dit snel en eenvoudig en de betaling registreren. Op het einde van de dag wordt deze GUI afgesloten en wordt het dagrapport gegenereerd.

- Reporting API

  Deze API wordt gebruikt om de dagrapporten te genereren wanneer een kassasysteem afgesloten worden. Na het genereren van het dagrapport wordt er door deze API een push-notificatie uitgestuurd naar de Reporting applicatie om de gebruiker te laten weten dat er een nieuw dagrapport beschikbaar is.

- Reporting Client

  Deze applicatie laat toe om de data aangeleverd door de dagrapporten te visualiseren in verschillende grafieken te kunnen bekijken en zo een overzicht te verschaffen hoe de winkels het op vlak van omzet doen.

5

Conclusie

Inleiding

In dit hoofdstuk zal er getracht worden een conclusie te formuleren op de hoofd-onderzoeksvraag en diens subvragen.

5.1. Subonderzoeksvraag 1

Na het opstellen van de proof-of-concept werden er metingen uitgevoerd op zowel het bestaande systeem als de proof-of-concept. Dit met als doel een conclusie op de eerste subonderzoeksvraag, namelijk: “Wat is de gemiddelde reactiesnelheid van de .NET MAUI applicatie en hoe verhoudt deze zich ten opzichte van de gemiddelde reactiesnelheid van het huidige systeem?”
De hypothese voor deze vraag luidt als volgt: “de gemiddelde reactietijd van de POC is kleiner dan de gemiddelde reactietijd van de huidige applicatie”
Om de hypothese te kunnen aannemen of verwerpen werden enkele statistische toetsen uitgevoerd:

1. Het gemiddelde van de twee datasets werd berekend

   - Voor het huidige systeem is het gemiddelde gelijk aan 1867,076
   - Voor de proof-of-concept is het gemiddelde gelijk aan 523,931

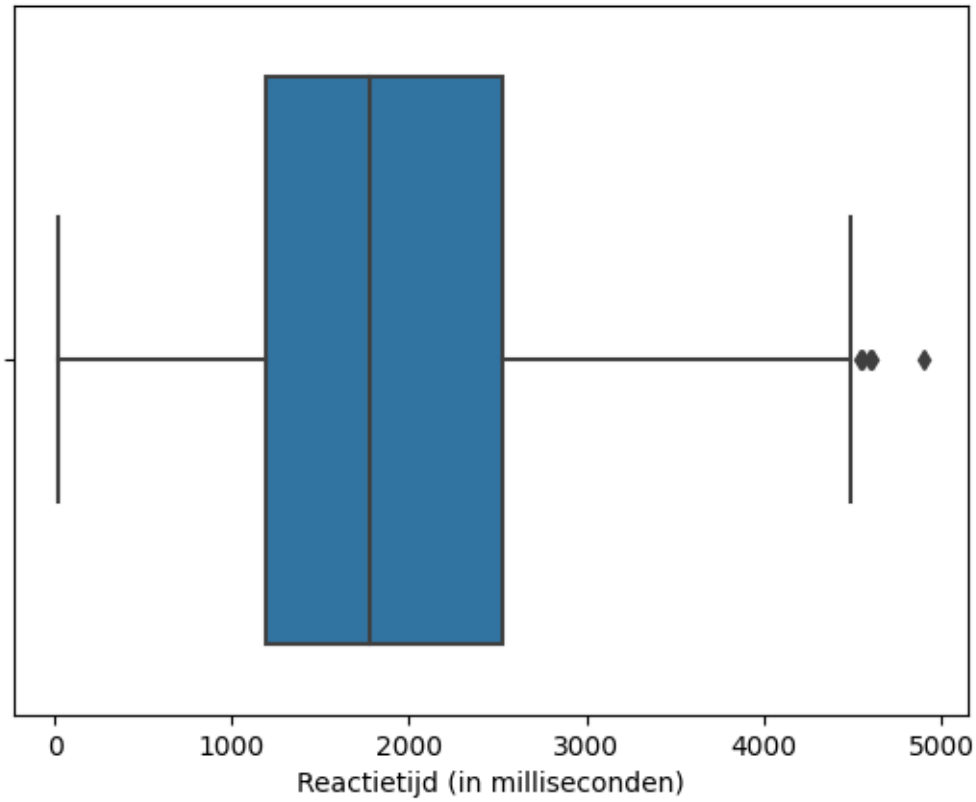
   Hieruit kan al geconcludeerd worden dat de gemiddelde reactiesnelheid van de proof-of-concept veel lager is dan de reactiesnelheid van het huidig systeem.

2. Een twee-sample T-test werd uitgevoerd. Deze test zou een definitief antwoord moeten geven over de geldigheid van de hypothese. Voor deze test werd een significantieniveau van 0,05 gebruikt. Daarnaast werden ook twee hypotheses opgesteld, namelijk:

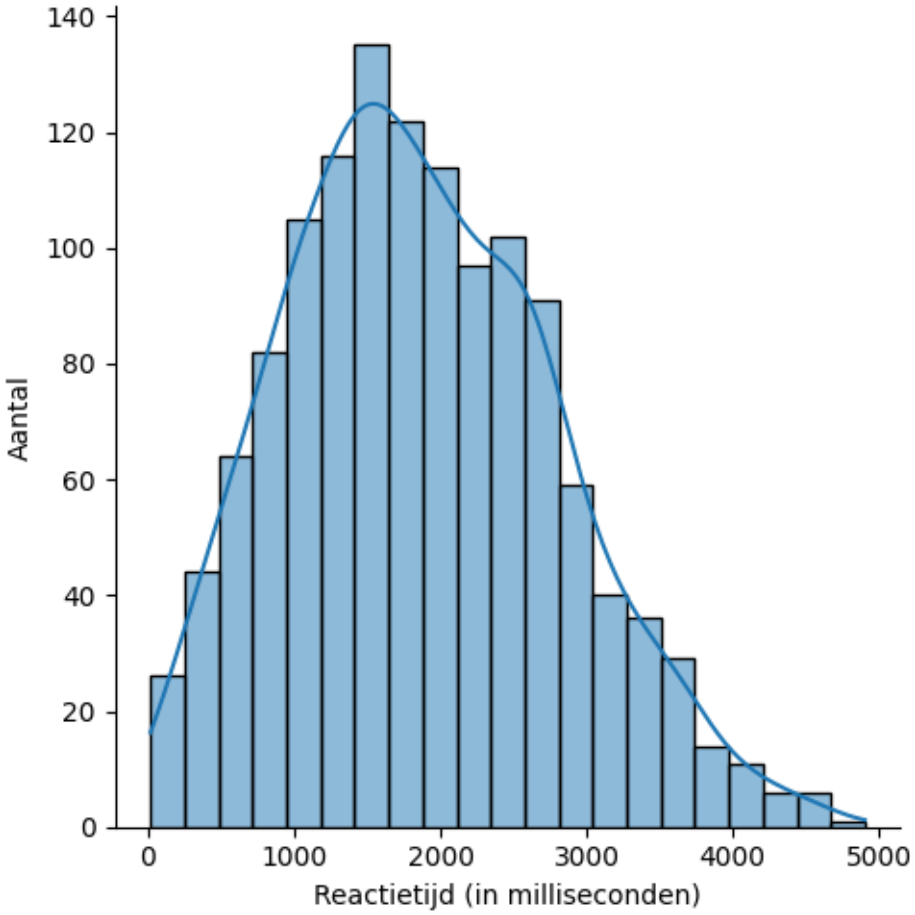
- H0: de gemiddelde reactietijd van de POC is groter of gelijk aan de gemiddelde reactietijd van de huidige applicatie
- H1: de gemiddelde reactietijd van de POC is kleiner dan de gemiddelde reactietijd van de huidige applicatie

Na het uitvoeren van deze test werd een p-waarde, ook wel de overschrijdingskans genoemd, vastgesteld die ongeveer gelijk was aan nul. Aangezien de p-waarde kleiner is dan het bovengenoemde significantieniveau, mag de nulhypothese verworpen worden.

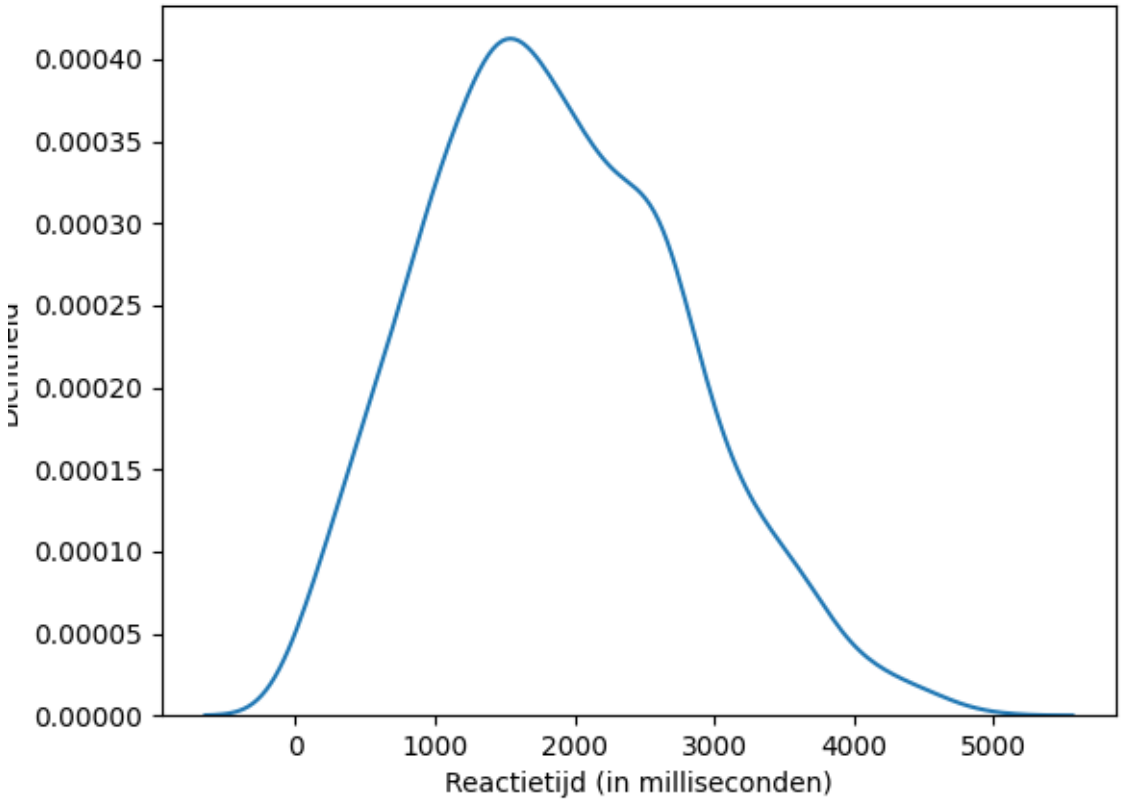
Kortom beide statistische proeven tonen aan dat de initiële hypothese klopt. Daarnaast kan er dankzij deze twee proeven ook een antwoord op de onderzoeksvraag geformuleerd worden, namelijk: de gemiddelde reactiesnelheid van de .NET MAUI-applicatie is gelijk aan 523,931 en is in verhouding tot het huidig systeem 3,56 keer zo snel op vlak van reactietijd.



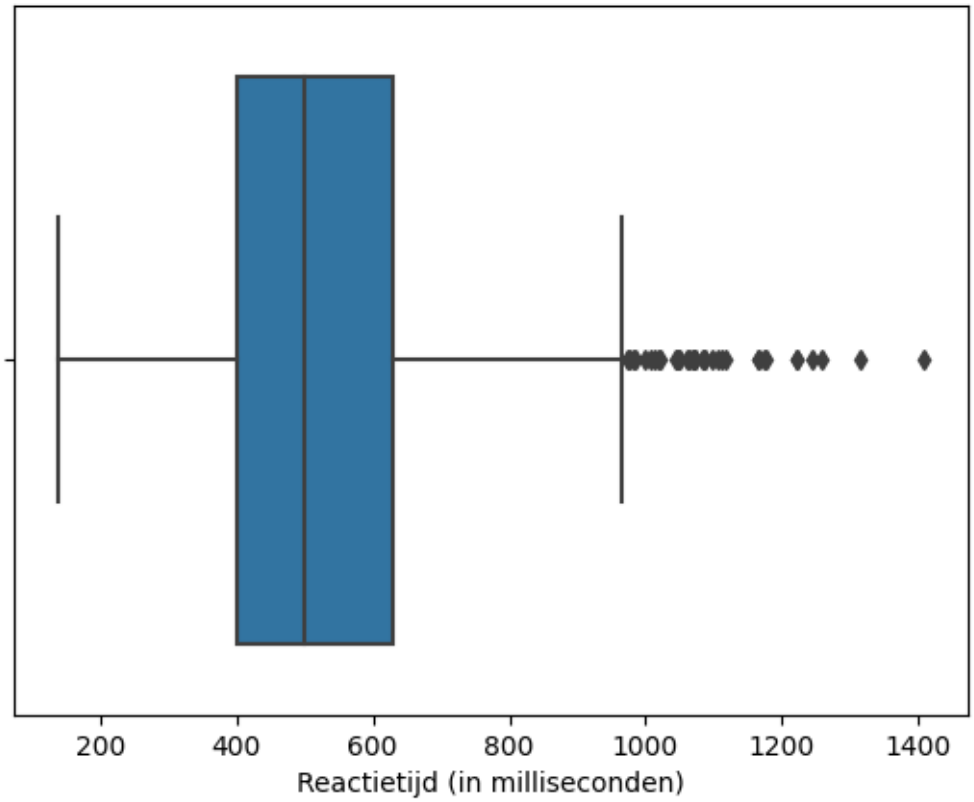
Figuur (5.1)
Boxplot van de reactietijden in het huidige syteem



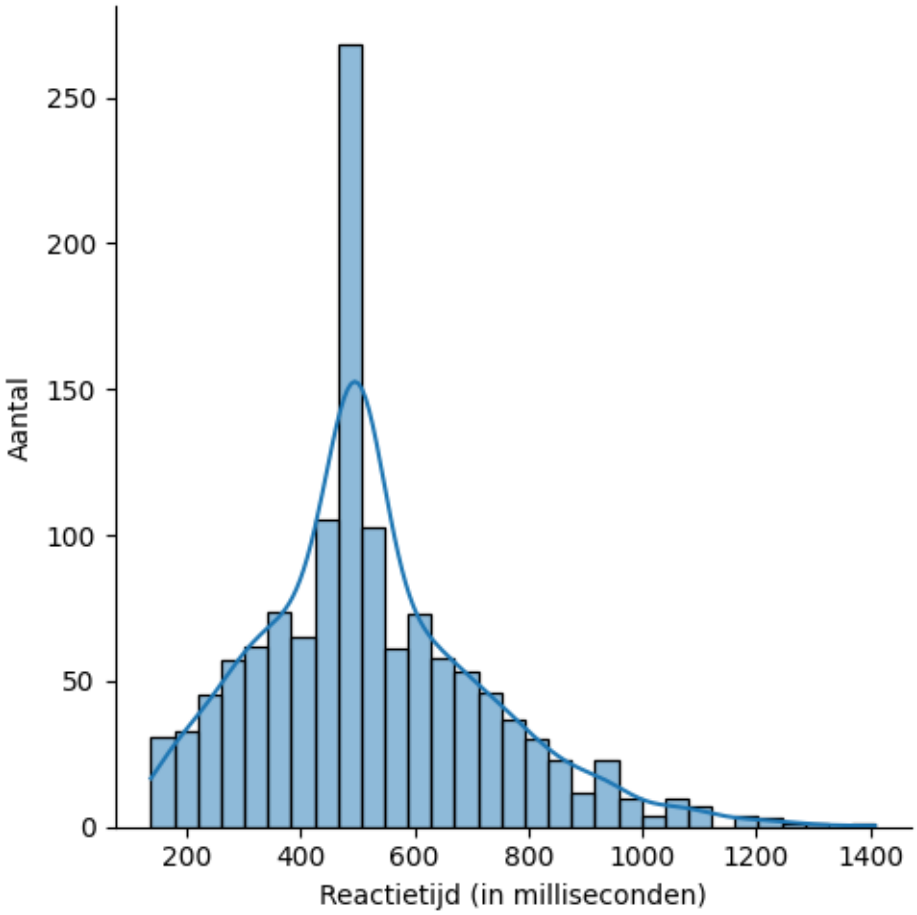
Figuur (5.2)
Histogram van de reactietijden in het huidige syteem



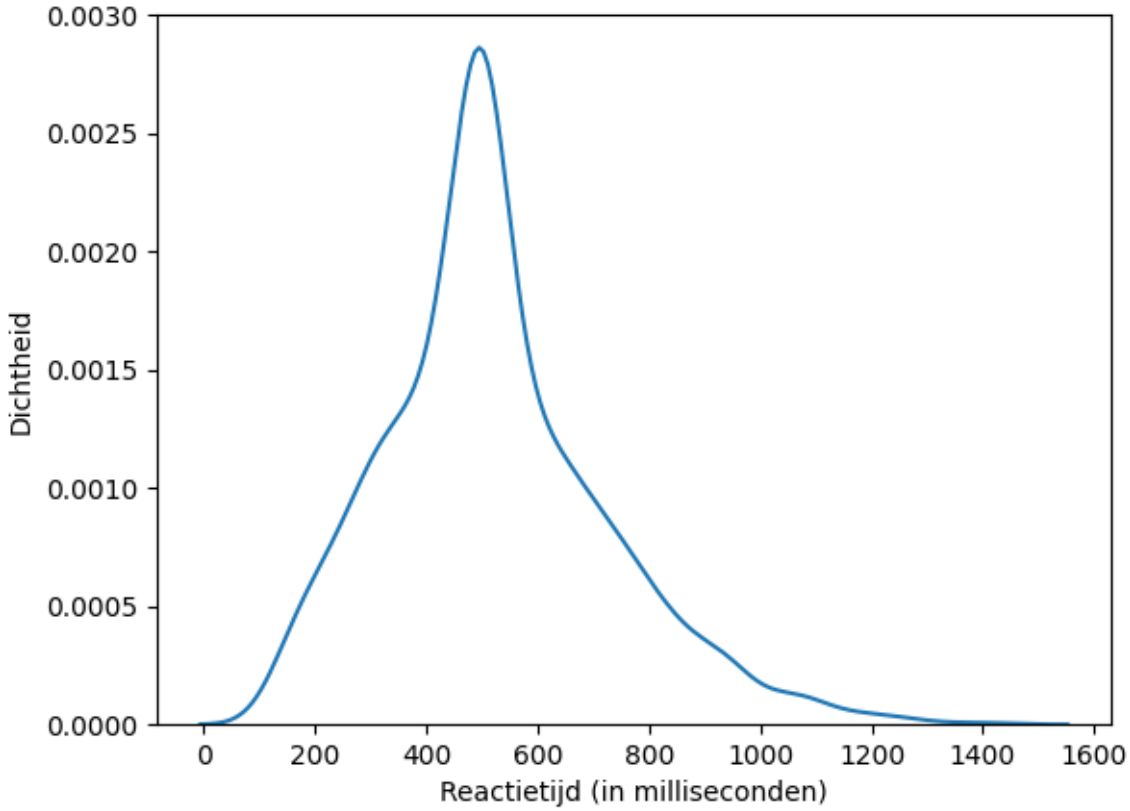
Figuur (5.3)
Densiteitsgrafiek van de reactietijden in het huidige sysyteem



Figuur (5.4)
Boxplot van de reactietijden in de proof-of-concept



Figuur (5.5)
Histogram van de reactietijden in de proof-of-concept



Figuur (5.6)
Densiteitsgrafiek van de reactietijden in de proof-of-concept

5.2. Subonderzoeksvraag 2

Het tweede luik van de onderzoeksvraag luidt als volgt: “Laat .NET MAUI het toe om push-notificatie te versturen?”
.NET MAUI levert geen algemene bibliotheek aan om push-notificaties te implementeren in de mobiele applicaties. Om dit te kunnen verwezenlijken, moeten we voor ieder platform een verschillend project opzetten die enkel de applicatie voor dat platform kan gaan bouwen. Daarnaast is er ook nood aan het gebruik van een bibliotheek uitgegeven door een derde partij. In dit geval gaat het om de Plugin.Firebase biblitoheek gemaakt en onderhouden door Tobias Buchholz. Dit zorgt ervoor dat de applicatie moet steunen op Firebase, wat een verzameling is van backend cloud computing services aangeboden door Google.
Dit alles zorgt ervoor dat er geconludeerd kan worden dat .NET MAUI het wel toelaat om push-notificaties te versturen, maar dat dit niet eenmalig geïmplementeerd kan worden voor de drie platformen samen. Dit zorgt er dan ook voor dat enkele van de voordelen van een cross-platform framework wegvallen.

5.3. Algemene conclusie

Door de conclussies geformuleerd voor de twee subonderzoeksvragen kan nu ook een algemeen besluit gevormd worden. Zoals in de inleiding wordt vermeld is het onderzoek een partiëel succes wanneer een positief besluit gevormd kan worden voor de eerste subonderzoeksvraag. Deze proef is een compleet succes wanneer de tweede subonderzoeksvraag een positief resultaat verkrijgt.
Dit onderzoek is alvast een partiëel succes aangezien het antwoord op de vraag: "Wat is de gemiddelde reactiesnelheid van de .NET MAUI applicatie en hoe verhoudt deze zich ten opzichte van de gemiddelde reactiesnelheid van het huidige systeem?" als volgt luidt: "De gemiddelde reactiesnelheid van .NET MAUI is gelijk aan 523,931 en is in verhouding tot de gemiddelde reactiesnelheid van het huidige systeem 3,56 keer zo snel".
Over het complete succes van de proef kan gedebateerd worden aangezien het mogelijk is om push-notificaties te versturen. Echter is dit momenteel enkel mogelijk indien er voor de verschillende platformen een aparte codebase wordt gebouwd (wat indruist tegen de visie van .NET MAUI) en er geen standaard bibliotheken aangeleverd worden door Microsoft. In deze proef wordt dit dan ook niet aanzien als een succes.
Dit alles zorgt er voor dat de algemene conclusie van deze proef als volgt luidt: .NET MAUI is voldoende matuur om een POS-systeem geschreven in COBOL te vervangen. Echter wanneer nood is aan het gebruik van niche features, zoals bijvoorbeeld het gebruik van push-notificaties, heeft .NET MAUI nog wat zaken die (beter) geïmplementeerd moeten worden.

A

Onderzoeksvoorstel

Het onderwerp van deze bachelorproef is gebaseerd op een onderzoeksvoorstel dat vooraf werd beoordeeld door de promotor. Dat voorstel is opgenomen in deze bijlage.

Samenvatting

Native applicatie development is een tijdrovend en duur proces waarbij een applicatie ontwikkeld wordt voor verschillende platformen, zoals iOS en Android. Hier kan cross-platform development een goede oplossing bieden. Cross-platform laat toe om met één codebase in één en dezelfde programmeertaal een applicatie te ontwikkelen die op de meeste gebruikte platformen kan draaien. In deze bachelorproef zal er worden nagegaan of .NET MAUI al voldoende ontwikkeld is om een real-time kassa-applicatie, geschreven in COBOL, te vervangen met een productiewaardige cross-platform applicatie ontwikkeld in .NET. In de eerste fase van het onderzoek zal er nagegaan worden of het analoge signaal van een printer opgevangen en omgevormd kan worden met een Raspberry Pi. In de tweede fase zal er een real-time .NET MAUI-applicatie ontwikkeld worden die steeds kan worden geüpdate wanneer er een signaal van een printer binnenkomt. Tenslotte zal de .NET MAUI-applicatie gecompileerd worden naar een mobiel platform. Hier zal nagegaan worden of dit zonder enige probleem kan. Er wordt verwacht dat de .NET MAUI-applicatie sneller zal reageren dan de huidige COBOL-applicatie. Daarnaast wordt er ook aangenomen dat de user experience al zodanig ontwikkeld is dat er slechts kleine problemen zouden voorkomen bij het compileren naar een mobiel platform. De beoogde doelgroep voor dit onderzoek is elke ontwikkelaar die de overweging maakt om .NET MAUI te gebruiken om een cross-platform applicatie te ontwikkelen.

A.1. Introductie

Een bedrijf (Goossens NV) heeft enkele copycentra, waar ze momenteel nog steeds werken met een kassasysteem dat geschreven is in COBOL. Op het einde van de dag wanneer de winkel sluit, moet steeds een dagrapport worden gemaakt en deze worden dan op een floppydisk geplaatst. Aan het begin van afgelopen zomer stoot het bedrijf op het probleem dat er geen floppydisks meer geproduceerd worden. Dit heeft als gevolg dat de dagrapporten niet meer bewaard kunnen worden. In dit onderzoek zal worden nagegaan of .NET MAUI, een cross-platform programmeertaal, al voldoende ontwikkeld is om een nieuw real-time kassasysteem op te zetten.

A.2. State-of-the-art

A.2.1. Cross-platform development

Cross-platform development, ook wel multi-platform development genoemd, is een manier van ontwikkelen dat toelaat om met één programmeertaal en bijgevolg één codebase een applicatie te ontwikkelen die op verschillende platformen kan draaien (Kotlin Foundation, 2022).

Native development VS cross-platform development

De term “native development” geeft aan dat een applicatie uitsluitend ontwikkeld wordt voor één bepaald platform, bijvoorbeeld iOS. De applicatie wordt geprogrammeerd met de door dat platform voorziene programmeertaal en hulpmiddelen (Marchuk, g.d.).
Om native Android-applicaties te schrijven, moet de developer gebruik maken van Java of Kotlin. De talen, die dan weer native ondersteund worden door het iOS platform, zijn Objective-C en Swift (Schmitt, 2022).

.NET MAUI

.NET MAUI is de uitgebreide evolutie van Xamarin.Forms om niet alleen mobiele applicaties te ontwikkelen, maar ook desktopapplicaties. .NET MAUI verenigt Android, iOS, macOS en Windows API’s in een enkele API die het toelaat om met één codebase applicaties te ontwikkelen voor de verschillende platformen (Britch & Gechev, 2022).

A.2.2. Real-time applicaties

De term real-time applicatie wordt door Lutkevich (2022) als volgt gedefinieerd: “Real-time-toepassingen zijn toepassingen die binnen een onmiddellijk tijdskader werken; zij detecteren, analyseren en handelen op streaming gegevens terwijl die zich voordoen. Dit in tegenstelling tot een database-gerichte toepassing waarbij informatie wordt opgenomen en opgeslagen in een database (in de cloud of op locatie) voor toekomstige analyse.”

Om de werking van real-time applicaties te kunnen verduidelijken, moet het websocket protocol uitgelegd worden.

HyperText Transfer Protocol

HyperText Transfer Protocol, ook wel gekend als HTTP, is het standaard protocol dat webbrowsers gebruiken om informatie op te vragen bij de server. HTTP werkt volgens het client-server principe. Dit houdt in dat de client, in de meeste gevallen de webbrowser, steeds een dataverzoek stuurt naar de server. Deze zal op zijn beurt dan de gewenste data verzamelen en terugsturen als een document van een bepaald type. Dit kan bijvoorbeeld gaan over HTML-documenten voor webpagina's of JSON-documenten om data te transfereren (MDN, 2022).
Zoals hierboven beschreven is HTTP unidirectioneel. Dit wil zeggen dat de dataaanvragen enkel en alleen kunnen worden opgestart door een client en kunnen alleen beantwoord worden door de server. Na versturen van het antwoord, wordt de verbinding verbroken (MDN, 2022).

WebSocket

WebSocket is een bidirectioneel protocol die dezelfde noden vervult als HTTP, maar in tegenstelling tot HTTP is WebSocket een stateful protocol. Hiermee wordt bedoeld dat de verbinding tussen de client en de server in stand wordt gehouden ook al is de data aanvraag verwerkt. De enige manier om de verbinding te verbreken is wanneer één van de twee partijen besluit om de verbinding stop te zetten. Een groot verschil met HTTP is ook dat de communicatie bij WebSocket bidirectioneel is, wat wil zeggen dat de client de server kan aanspreken, maar ook dat de server de client kan aanspreken (GeeksforGeeks, 2022).

A.3. Methodologie

Het huidige kassasysteem werkt als volgt: een klant komt binnen in één van de copy centra en krijgt een printer aangewezen door de verko(o)p(st)er die op dat moment in de winkel staat. Eens de klant het printproces gestart heeft, wordt er per geprinte pagina een klik doorgestuurd naar een elektronisch bord. Dit bord weet op welke interface de klik binnengekomen is, welke printer het signaal verstuurd heeft en vertaalt dit dan naar een digitaal signaal dat verstuurd wordt naar het kassasysteem. Eens het signaal is aangekomen, wordt de GUI geüpdatet zodat op het einde van het printproces de verkoopster exact weet hoeveel pagina's er geprint zijn en hoeveel het te betalen totaal is.
Hieruit kunnen volgende onderzoeksvragen naar voor geschoven worden:

1. Is .NET MAUI voldoende ontwikkeld om het bestaande kassasysteem te vervangen?
2. Is het mogelijk om de desktop applicatie te vertalen naar een mobiel platform?

3. Is de ontwikkelde .NET MAUI-applicatie sneller in het verwerken van de printerklik dan de huidige applicatie.

Om na te gaan of .NET MAUI al voldoende ontwikkeld is voor deze toepassing zal er een proof-of-concept opgezet worden. In de eerste fase zal er uitgezocht worden of het analoge signaal, de zogenaamde klik, kan worden opgevangen door een Raspberry Pi en kan worden vertaald naar een digitaal signaal.
In het tweede luik van het onderzoek zal er een .NET MAUI-applicatie worden ontwikkeld die het huidige COBOL-kassasysteem moet vervangen. Deze applicatie moet zichzelf steeds kunnen updaten wanneer er een klik binnenkomt van de printer. Om dit te kunnen verwezenlijken zal er ook een API (Application Programming Interface) opgezet worden. Deze API moet het mogelijk maken om door de Raspberry Pi aangesproken te worden elke keer dat er een klik van een printer binnenkomt. Daarnaast moet de API ook de client kunnen updaten na elke klik. Deze feature zal dan ook geïmplementeerd worden met behulp van SignalR. Een .NET library die het toelaat om real-time applicatie op te zetten. SignalR maakt het mogelijk om op basis van websockets bidirectioneel te communiceren tussen de client en de server. Met andere woorden de client kan data opvragen aan de server indien nodig, maar de server kan ook zonder aanvraag van de client data doorgeven. De bidirectionele communicatie moet verhelpen dat de client steeds na een aantal seconden aan de server moet vragen of er al nieuwe kliks zijn binnengekomen.
In de laatste fase zal er worden nagegaan of de applicaties vertaald kan worden naar een Android-applicatie. Deze zou het mogelijk moeten maken om op het einde van de dag een pushmelding te sturen zodat de dagrapporten van de copycentra opgehaald en bekeken kunnen worden.
Om te weten of .NET MAUI voldoende ontwikkeld is om de huidige applicatie te vervangen, zal er een meting gebeuren. De meting in kwestie zal starten vanaf het moment dat de printer één klik verstuurt en stopt op het moment dat de grafische interface zichzelf update. Daarnaast zal er ook gekeken worden naar developer experience bij het ontwikkelen in .NET MAUI. Hier wordt er nagegaan of de ”intellisense”, een helper tool die suggesties geeft tijdens het programmeren, goed werkt. Met andere woorden; worden de juiste suggesties gegeven tijdens het programmeren. Ook zal er gekeken worden of er nog integratiebugs in Visual Studio zitten: krijg je foutmeldingen van Visual Studio die niets te maken hebben met de code, maar met de .NET MAUI integratie etc.

A.4. Verwacht resultaten

De metingen zouden moeten uitwijzen dat de .NET MAUI een grote stap vooruit is in vergelijking met de huidige kassa-applicatie geschreven in COBOL. De .NET applicatie zou sneller moeten reageren dan de huidige applicatie.
Daarnaast zou de developer experience al zodanig gevorderd zijn dat er slechts

kleine bugs overblijven in de integratie van .NET MAUI en Visual Studio. Het zou mogelijk moeten zijn om .NET MAUI te gebruiken om een productiewaardige applicatie te ontwikkelen.

A.5. Verwachte conclusie

.NET MAUI is een product dat nog steeds volop in ontwikkeling is. Dit kan er dan ook voor zorgen dat de developer experience nog niet optimaal is: de “intellisense” geeft geen of foute suggesties, Visual Studio geeft foutmeldingen die niets te maken hebben met de geschreven code, de applicatie crasht soms vanzelf etc. Als men toch wenst .NET MAUI reeds te gebruiken, zou dit echter wel mogelijk moeten zijn. Men moet enkel opletten bij het gebruik van bepaalde features en packages. Desondanks de kleine ongemakken zou .NET MAUI toch de toekomst kunnen zijn van cross-platform development waar kleinere development-teams, die een applicatie moeten ontwikkelen voor meerdere platformen, gretig gebruik van kunnen maken.

Bibliografie

Ab, S. (g.d.). *The current HTTP/2 adoption and the quite recent history*. https://www.shimmercat.com/blog/http2adoption/

Abby. (2023, februari). *COBOL Guide: History, Origin, and More*. https://history-computer.com/cobol-guide/

Ably. (2020, juli). *WebSockets - A Conceptual Deep Dive*. Ably. https://ably.com/topic/websockets

Brady Gaster. (2020, september). *Introduction to SignalR*. Microsoft. https://learn.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr

Britch, D. (2023, januari). *What is .NET MAUI?* https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-7.0

Britch, D., & Gechev, I. (2022, november 8). *What is .NET MAUI?* https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-7.0

ByteByteGo. (2022, december). What is OSI Model | Real World Examples. https://www.youtube.com/watch?v=0y6FtKsg6J4

Cloudflare. (g.d.). *What is HTTP?* Cloudflare. https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/

Combell. (g.d.). *HTTP/2 and HTTP/3 | Operation and benefits*. https://www.combell.com/en/technology/http

Fette, I., & Melnikov, A. (2011). The WebSocket Protocol. *Internet Engineering Task Force*. https://doi.org/10.17487/rfc6455

Fulber-Garcia, V. (2022, november). *HTTP: 1.0 vs. 1.1 vs 2.0 vs. 3.0*. https://www.baeldung.com/cs/http-versions

GeeksforGeeks. (2022, februari 21). *What is web socket and how it is different from the HTTP?* https://www.geeksforgeeks.org/what-is-web-socket-and-how-it-is-different-from-the-http/

Grigorik, I. (g.d.). *High Performance Browser Networking*. https://www.oreilly.com/library/view/high-performance-browser/9781449344757/ch09.html

Grigorik, I. (2016, september). *Introduction to HTTP/2*. https://web.dev/performance-http2

Hesham, M. (2019, augustus). *HTTP/1.1 Introduction*. https://medium.com/@he4am3id66/http-1-1-introduction-76d90b4834bf

Kathiresan, S. G. (2022, november). *Xamarin Versus .NET MAUI*. https://www.syncfusion.com/blogs/post/xamarin-versus-net-maui.aspx

Koleva, P. (2023, februari). *From Xamarin to MAUI, What has changed?* https://flatrocktech.com/xamarin-forms-maui-migration/

Kotlin Foundation. (2022, september 6). *What is cross-platform mobile development?* Kotlin Foundation. https://kotlinlang.org/docs/cross-platform-mobile-development.html

Kotlin Foundation. (2023, maart 29). *Native and cross-platform app development: how to choose?* Kotlin Foundation. https://kotlinlang.org/docs/native-and-cross-platform.html

Lutkevich, B. (2022, mei 9). *real-time application (RTA)*. https://www.techtarget.com/searchunifiedcommunications/definition/real-time-application-RTA

Marchuk, A. (g.d.). *Native vs Cross-Platform Development: Pros and Cons Revealed*. https://www.uptech.team/blog/native-vs-cross-platform-app-development

MDN. (2022, november 22). *An overview of HTTP*. https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

MDN. (2023a, maart). *An overview of HTTP*. MDN. https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

MDN. (2023b, maart). *Evolution of HTTP*. MDN. https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP

Micro Focus. (2022, maart). *COBOL Market Shown to be Three Times Larger than Previously Estimated in New Independent Survey*. Micro Focus. https://www.microfocus.com/en-us/press-room/press-releases/2022/cobol-market-shown-to-be-three-times-larger-than-previously-estimated-in-new-independent-survey

National Museum of American History. (2013, september). *Proposing COBOL*. National Museum of American History. https://americanhistory.si.edu/cobol/proposing-cobol

Paessler. (g.d.). *What is HTTP?* Paessler. https://www.paessler.com/it-explained/http

Ramel, D. (2022, augustus). *What's Next for .NET MAUI? Roadmap & Xamarin Sunset Unveiled*. https://visualstudiomagazine.com/articles/2022/08/22/net-maui-roadmap.aspx

Reuters. (2017). *COBOL blues*. http://fingfx.thomsonreuters.com/gfx/rngs/USA-BANKS-COBOL/010040KH18J/index.html

Schmitt, J. (2022, augustus 24). *Native vs cross-platform mobile app development*. https://circleci.com/blog/native-vs-cross-platform-mobile-dev/?utm_source=google

Singh, S. (g.d.). *What is HTTP Long Polling ?* https://www.educative.io/answers/what-is-http-long-polling

Suse. (g.d.). *What is Real-Time?* https://www.suse.com/suse-defines/definition/real-time/

UXDivers. (g.d.). .*NET MAUI vs Xamarin.Forms: A Comparison of Cross-Platform Frameworks*. https://grialkit.com/blog/learn-the-key-differences-between-net-maui-vs-xamarin-forms-for-cross-platform-mobile-and-desktop-development