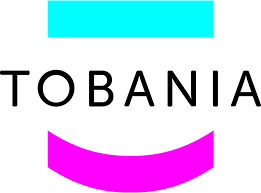


**Automatiseringstools**

Joren Vanderzande

Promotoren:

|  |  |
| --- | --- |
| Kris Vanreyten | Tobania |
| Carine Derkoningen | Hogeschool PXL Hasselt |



**Automatiseringstools**

Joren Vanderzande

Promotoren:

|  |  |
| --- | --- |
| Kris Vanreyten | Tobania |
| Carine Derkoningen | Hogeschool PXL Hasselt |

Dankwoord

Deze bachelorpaper had ik nooit zonder de hulp van anderen kunnen realiseren. Daarom wil ik graag mijn dankbaarheid uiten naar de mensen die mij geholpen en gesteund hebben.

Eerst en vooral wil ik graag mijn lectoren bedanken die mij een stevige basis van programmeren en andere onderwerpen hebben gegeven, in het bijzonder mijn hogeschoolpromotor, Carine Derkoningen. Zij stond altijd klaar om mijn teksten te verbeteren en advies te geven. Ook was het fijn om iedere vrijdagavond, net voor het weekend, even een gesprekje te hebben over hoe de week verlopen was. Dit was altijd wel een moment waar ik naar uitkeek.

Vervolgens wil ik ook mijn stagebedrijf, Tobania, heel erg bedanken. Ze hebben mij de kans gegeven om te bewijzen wat ik waard ben. Vooral mijn bedrijfspromotor, Kris Vanreyten, zou ik willen bedanken om altijd zo vriendelijk en begripvol te zijn. Hij stond open voor mijn vragen en gaf mij veel praktische hulp. Ook David Nysten mag ik niet vergeten. Hij hielp mij regelmatig op zoek te gaan naar de juiste oplossing.

Tijdens mijn stageopdracht had ik veel contact met de klant, Acerta. Ik wil Ruud Boyden bedanken om ondanks zijn drukke werkschema toch tijd vrij te maken voor mij.

Ten slotte wil ik graag mijn vrienden, familie en vooral mijn ouders bedanken voor hun steun tijdens deze periode.

Abstract

Tobania is een dynamisch, Belgisch Business & *Technology* Consulting-bedrijf met meer dan duizend betrokken Tobians. Dit zorgt voor een brede waaier aan services die Tobania aanbiedt aan bedrijven en overheden, waaronder Finance, Design, Development, Testing, Support en Agile. Voor elke service is er een apart departement voorzien. Tobania werkt voor grote bedrijven, zoals Acerta, NMBS, BNP Paribas Fortis en voor grote kmo’s. De grote bedrijven zijn de trendsetters en de kmo’s volgen. Hun motto is *“*Your wingman making digital work*”.* Dit verwijst naar het consultantproces.

Tijdens de stage wordt een applicatie ontwikkeld voor een klant van Tobania, namelijk Acerta. Deze applicatie moet achtergrondtaken starten op gewenste tijdstippen in Arno, de reeds bestaande software van Acerta. Verder zou de applicatie de feedback van deze taken moeten kunnen lezen en opslaan in een log. Dit werd in het verleden gedaan met Windows Taakplanner maar deze applicatie is niet gebruiksvriendelijk.

De applicatie wordt ontwikkeld in .NET Core 3.1. Voor het automatisch uitvoeren van taken wordt het Hangfire-framework gebruikt. Dit framework wordt geconfigureerd in de applicatie en voert de taken op de achtergrond uit. Dit zorgt ervoor dat de applicatie niet zal vastlopen. Om de applicatie *lightweight* te houden, worden de taken opgeslagen in een SQLite-database. Voor de frontend wordt Windows Presentation Foundation (WPF) gebruikt. Dit framework geeft veel vrijheid bij het maken van de frontend. Hierdoor is het mogelijk om de applicatie er modern uit te laten zien. Ook wordt Prism gebruikt om het toepassen van Model-View-ViewModel (MVVM) gemakkelijker te maken.

Er is een onderzoek gekoppeld aan deze stage. Voor dit onderzoek is het de bedoeling om uit te zoeken of Hangfire wel het beste framework is om deze module te maken. Dit gebeurt door onderzoek te doen naar welke andere technologieën hiervoor gebruikt kunnen worden. Vervolgens wordt er een lijst gemaakt van de voor- en nadelen van deze technologieën. Ook wordt er een Proof of Concept (PoC) gemaakt met iedere tool/framework om na te gaan wat de mogelijkheden zijn.

Tijdens het onderzoek is gebleken dat Hangfire de beste oplossing is. Omdat het voldoet aan alle criteria en gemakkelijk is om te leren. Quartz.NET is zeker geen slechte optie. Ontwikkelaars die al werken met Quartz.NET moeten zeker niet overgaan naar Hangfire. Azure Logic Apps daarentegen is geen goede oplossing omdat het niet gebruiksvriendelijk is voor mensen met weinig computerkennis.

Over het algemeen is de stage goed verlopen. Er is een tool ontwikkeld die automatisch taken kan plannen. Deze tool is gebruiksvriendelijk en geeft feedback van de uitgevoerde taken. De code is gestructureerd en er werd een code coverage behaald van meer dan 80% maar nog belangrijker, de klant is tevreden.

Inhoudsopgave

[Dankwoord ii](#_Toc74485328)

[Abstract iii](#_Toc74485329)

[Inhoudsopgave iv](#_Toc74485330)

[Lijst van gebruikte figuren vii](#_Toc74485331)

[Lijst van gebruikte tabellen viii](#_Toc74485332)

[Lijst van gebruikte afkortingen ix](#_Toc74485333)

[Inleiding 1](#_Toc74485334)

[I. Stageverslag 2](#_Toc74485335)

[1 Bedrijfsvoorstelling 2](#_Toc74485336)

[1.1 Situering 2](#_Toc74485337)

[1.2 Tobania.Development 2](#_Toc74485338)

[1.1 Acerta 2](#_Toc74485339)

[2 Voorstelling stageopdracht 3](#_Toc74485340)

[2.1 Automatiseringstool 3](#_Toc74485341)

[2.1.1 Probleemstelling 3](#_Toc74485342)

[2.1.2 Doelstelling 3](#_Toc74485343)

[2.2 Windows Taakplanner 4](#_Toc74485344)

[3 Uitwerking stageopdracht 5](#_Toc74485345)

[3.1 Informatie-uitwisseling 5](#_Toc74485346)

[3.2 Voorkomen van problemen 5](#_Toc74485347)

[3.2.1 Unittesten 5](#_Toc74485348)

[3.2.2 Code coverage 5](#_Toc74485349)

[3.3 Gebruikte technologieën 6](#_Toc74485350)

[3.3.1 .NET Core 6](#_Toc74485351)

[3.3.2 Windows Presentation Foundation 6](#_Toc74485352)

[3.3.3 Entity Framework 6](#_Toc74485353)

[3.3.4 Hangfire 7](#_Toc74485354)

[3.3.5 Topshelf 7](#_Toc74485355)

[3.3.6 Serilog 7](#_Toc74485356)

[3.3.7 Prism 7](#_Toc74485357)

[3.3.8 NUnit 7](#_Toc74485358)

[3.3.9 Moq 8](#_Toc74485359)

[3.4 Planning 8](#_Toc74485360)

[3.5 Analyse 9](#_Toc74485361)

[3.5.1 Userstory’s 9](#_Toc74485362)

[3.5.2 Proof of Concept 10](#_Toc74485363)

[3.6 Ontwerp 11](#_Toc74485364)

[3.6.1 Architectuurschema 11](#_Toc74485365)

[3.6.2 User experience design 12](#_Toc74485366)

[3.6.3 Wireframes en prototypes 13](#_Toc74485367)

[3.7 Ontwikkeling 20](#_Toc74485368)

[3.7.1 Projectstructuur 20](#_Toc74485369)

[3.7.2 Integratie 22](#_Toc74485370)

[3.7.3 Eindresultaat 23](#_Toc74485371)

[3.8 Testing 30](#_Toc74485372)

[3.8.1 Façadepatroon 30](#_Toc74485373)

[3.8.2 Code coverage 30](#_Toc74485374)

[4 Extra stageopdracht: Talkdesk Proof of Concept 31](#_Toc74485375)

[4.1 Wat is Talkdesk 31](#_Toc74485376)

[4.2 Doelstelling 31](#_Toc74485377)

[4.3 Uitwerking extra opdracht 31](#_Toc74485378)

[4.3.1 Authenticatie 31](#_Toc74485379)

[4.3.2 Reporting API 31](#_Toc74485380)

[4.3.3 Backend 32](#_Toc74485381)

[4.3.4 Frontend 32](#_Toc74485382)

[4.3.5 Conclusie 32](#_Toc74485383)

[II. Onderzoekstopic 33](#_Toc74485384)

[1 Vraagstelling onderzoek 33](#_Toc74485385)

[2 Onderzoeksmethode 33](#_Toc74485386)

[3 Technologieën 34](#_Toc74485387)

[3.1 Hangfire 34](#_Toc74485388)

[3.2 Quartz.NET 34](#_Toc74485389)

[3.3 Azure Logic Apps 35](#_Toc74485390)

[4 Literatuurstudie 36](#_Toc74485391)

[4.1 Hangfire 36](#_Toc74485392)

[4.1.1 Dashboard 36](#_Toc74485393)

[4.1.2 Soorten taken 36](#_Toc74485394)

[4.1.3 Documentatie 36](#_Toc74485395)

[4.1.4 Windows-service 36](#_Toc74485396)

[4.1.5 Opslagmogelijkheden 37](#_Toc74485397)

[4.1.6 Prijs 37](#_Toc74485398)

[4.1.7 NuGet 37](#_Toc74485399)

[4.2 Quartz.NET 37](#_Toc74485400)

[4.2.1 Taken, triggers en planners 37](#_Toc74485401)

[4.2.2 Prijs 37](#_Toc74485402)

[4.2.3 Complexiteit 38](#_Toc74485403)

[4.2.4 Opslagmogelijkheden 38](#_Toc74485404)

[4.3 Azure Logic Apps 38](#_Toc74485405)

[4.3.1 Triggers, acties en connectors 38](#_Toc74485406)

[4.3.2 Design tool 39](#_Toc74485407)

[4.3.3 Prijs 39](#_Toc74485408)

[4.3.4 Uitbreidbaarheid 39](#_Toc74485409)

[5 Proof of Concept 40](#_Toc74485410)

[5.1 Hangfire 40](#_Toc74485411)

[5.1.1 Installatie 40](#_Toc74485412)

[5.1.2 Configuratie 40](#_Toc74485413)

[5.1.3 Taak toevoegen 41](#_Toc74485414)

[5.2 Quartz.NET 41](#_Toc74485415)

[5.2.1 Installatie 41](#_Toc74485416)

[5.2.2 Configuratie 41](#_Toc74485417)

[5.2.3 Taak toevoegen 42](#_Toc74485418)

[5.3 Azure Logic Apps 42](#_Toc74485419)

[5.3.1 Registratie 42](#_Toc74485420)

[5.3.2 Voorbereiding 42](#_Toc74485421)

[5.3.3 Taak aanmaken 43](#_Toc74485422)

[6 Conclusie 44](#_Toc74485423)

[6.1 Voor- en nadelen 44](#_Toc74485424)

[6.1.1 Hangfire 44](#_Toc74485425)

[6.1.2 Quartz.NET 44](#_Toc74485426)

[6.1.3 Azure Logic Apps 44](#_Toc74485427)

[6.2 Welke tool/framework is de beste oplossing voor deze opdracht? 44](#_Toc74485428)

[III. Reflectie 45](#_Toc74485429)

[Bronnenlijst 47](#_Toc74485430)

Lijst van gebruikte figuren

[Figuur 1 Windows Taakplanner 4](#_Toc74485296)

[Figuur 2 Originele planning 8](#_Toc74485297)

[Figuur 3 Kanbanbord 9](#_Toc74485298)

[Figuur 4 Proof of Concept 10](#_Toc74485299)

[Figuur 5 Architectuurschema 11](#_Toc74485300)

[Figuur 6 Wireframe v\_1 Homepagina 13](#_Toc74485301)

[Figuur 7 Wireframe v\_1 Nieuwe taak pagina 14](#_Toc74485302)

[Figuur 8 Wireframe v\_2 Homepagina 15](#_Toc74485303)

[Figuur 9 Wireframe v\_3 Homepagina 16](#_Toc74485304)

[Figuur 10 Wireframe v\_3 Homepagina detail geslaagde taak 17](#_Toc74485305)

[Figuur 11 Wireframe v\_3 Homepagina detail afwachtende taak 17](#_Toc74485306)

[Figuur 12 Wireframe v\_3 Nieuwe taak pagina basistaak 18](#_Toc74485307)

[Figuur 13 Wireframe v\_3 Nieuwe taak pagina schema 18](#_Toc74485308)

[Figuur 14 Wireframe v\_3 Nieuwe taak pagina parameters 19](#_Toc74485309)

[Figuur 15 Projectstructuur 20](#_Toc74485310)

[Figuur 16 Homepagina resultaat 23](#_Toc74485311)

[Figuur 17 Homepagina met taak resultaat 24](#_Toc74485312)

[Figuur 18 Homepagina detail resultaat 24](#_Toc74485313)

[Figuur 19 Settingspagina resultaat 25](#_Toc74485314)

[Figuur 20 Job wizard basis taak resultaat 26](#_Toc74485315)

[Figuur 21 Job wizard dagelijks schema resultaat 26](#_Toc74485316)

[Figuur 22 Job wizard wekelijks schema resultaat 27](#_Toc74485317)

[Figuur 23 Job wizard maandelijks schema resultaat 27](#_Toc74485318)

[Figuur 24 Job wizard jaarlijks schema resultaat 28](#_Toc74485319)

[Figuur 25 Job wizard continuation-schema resultaat 28](#_Toc74485320)

[Figuur 26 Job wizard actie resultaat 29](#_Toc74485321)

[Figuur 27 Test coverage rapport 30](#_Toc74485322)

[Figuur 28 Talkdesk voorbeeld grafiek 32](#_Toc74485323)

[Figuur 29 Simpel architectuurschema [19] 34](#_Toc74485324)

[Figuur 30 Voorbeeld visuele designer [16] 35](#_Toc74485325)

[Figuur 31 Logic App met een schema als trigger en ping-commando als actie 43](#_Toc74485326)

[Figuur 32 TRIGGERcmd-Agent 43](#_Toc74485327)

Lijst van gebruikte tabellen

[Tabel 1 Voor- en nadelen Hangfire 44](#_Toc74485291)

[Tabel 2 Voor- en nadelen Quartz.NET 44](#_Toc74485292)

[Tabel 3 Voor- en nadelen Azure Logic Apps 44](#_Toc74485293)

Lijst van gebruikte afkortingen

|  |  |
| --- | --- |
| API | Application programming interface |
| CORS | Cross-Origin Resource Sharing |
| CPU | Central Processing Unit |
| CRM | Customer Relationship Management |
| CSS | Cascading Style Sheets |
| DI | Dependency injection |
| DTO | Data Transfer Object |
| GPU | Graphics processing unit |
| HTML | HyperText Markup Language |
| IBM | International Business Machines |
| IDE | Integrated Development Environment |
| IIS | Internet information services |
| JWT | JSON Web Token |
| MQ | Message Queue |
| MVP | Most Valuable Professional |
| MVVM | Model-View-ViewModel |
| ORM | Object-Relational Mapping |
| PoC | Proof of Concept |
| SAP | Systemen, Applicaties en Producten |
| WPF | Windows Presentation Foundation |
| XAML | eXtensible Application Markup Language |
| XML | Extensible Markup Language |

Inleiding

Ik heb Tobania leren kennen op het PXL-jobevent 2021. Het eerste wat mijn aandacht trok was dat het een bekend bedrijf was. Ik had er al van gehoord tijdens mijn periode op de PXL. Ook de programmeeromgeving sprak mij aan, ik programmeer namelijk het liefst in een .NET-omgeving. Ondanks dat de stage volledig online doorgaat, is het interessant dat Tobania een kantoor heeft op Corda Campus. Dat is bij mij in de buurt. In mijn achterhoofd hield ik al rekening met het feit dat ik er na mijn stage zou willen werken. Na het kennismakingsgesprek bleek dat het de meest interessante opdracht was.

Mijn stageopdracht is een project voor Acerta, een klant van Tobania. Acerta wil automatisch taken kunnen uitvoeren binnen Arno, hun bestaande softwarepakket. Op dit moment wordt dit aan de hand van Windows Taakplanner gedaan. Deze applicatie is totaal niet gebruiksvriendelijk voor mensen met weinig technische kennis. Hiervoor wordt een nieuwe applicatie ontwikkeld. Het ontwikkelproces van deze applicatie zal in het stageverslag worden toegelicht.

Hangfire wordt gebruikt in de applicatie, dit is niet het enige framework om taken automatisch uit te voeren in een .NET-omgeving. Was Hangfire wel het juiste framework? Deze vraag zal beantwoord worden in het onderzoek.

1. Stageverslag

# Bedrijfsvoorstelling

Eerst wordt er in het kort uitgelegd waar Tobania gesitueerd is. Daarna wordt er verder ingegaan op hoe de communicatie verloopt en hoe problemen vermeden worden binnen het ontwikkelingsproces.

## Situering

Tobania is een groot Belgisch consultant bedrijf binnen de ICT en Businesswereld. Tobania is ontstaan uit een fusie van Tobius en Saga Consulting Group op 2 december 2014. Tobania begeleidt zijn klanten in hun digitale bedrijfstransformatie. Zij bieden veel services aan, namelijk business analyse, ontwikkeling en testing. Ook bieden ze eigen specifieke softwareoplossing aan, zoals web development en personeelsplanning. “Met een omzet van 85 miljoen euro in 2018 en meer dan duizend IT-specialisten en businessconsultants (“Tobians”), behoort Tobania tot een van de grote spelers in België.” Het hoofdkantoor van Tobania is in Strombeek-Bever. Maar ze hebben ook kantoren in Gent en Hasselt. Omdat het een consultant bedrijf is, zitten veel van de medewerkers bij allerlei klanten in heel België. [1]

## Tobania.Development

Deze stageopdracht gaat door in de .Development afdeling van Tobania. Dit departement is gespecialiseerd in het maken van digitale applicaties. De services die dit departement aanbiedt zijn: applicatiemanagement, applicatie-integratie, DevOps-oplossingen, op maat gemaakte softwareontwikkeling en cloudarchitectuurontwikkeling. De .Development afdeling is opgedeeld in twee subgroepen. De eerste groep is het traditionele consultantproces waarbij de klant een aantal ontwikkelaars nodig heeft voor een bepaald project. Deze ontwikkelaars worden dan tewerkgesteld in een team van de klant maar werken nog altijd voor Tobania. De tweede groep noemen zij het “factorymodel”. In de plaats dat de klant een aantal ontwikkelaars nodig heeft, komt de klant deze keer met een projectvoorstel en de “factory” regelt zelf welke ontwikkelaar werkt aan dat project. Tobania heeft twee “factorymodels” het Java team in Brussel en een .NET team in Hasselt. Deze stageopdracht gaat specifiek door in het .NET team van het “factorymodel”.

## Acerta

Acerta, het bedrijf waarvoor deze stageopdracht gemaakt wordt, werd opgericht in 2001. Het is een *fullservice* humanresourcesdienstengroep. Het bedrijf heeft meer dan vijfentwintig kantoren over heel het land met een hoofdkantoor in Leuven. Zij ondersteunen ondernemers met een brede waaier aan advies, diensten en toepassingen rond loonadministratie, sociale zekerheid, vestigingsformaliteiten, enzovoort. Acerta helpt zowel starters, zelfstandigen, kmo’s als grote ondernemingen. Ze zijn bezig met zowel de private als de publieke sector. Daarnaast houden ze zich ook bezig met non-profit organisaties.

# Voorstelling stageopdracht

Zoals eerder reeds aangehaald wordt voor deze stageopdracht een tool ontwikkeld voor een klant van Tobania, namelijk Acerta.

## Automatiseringstool

### Probleemstelling

Acerta wil binnen zijn al bestaande software, Arno, taken kunnen plannen. Arno is een software die de gebruiker helpt met allerlei humanresourcestaken, een aantal voorbeelden hiervan zijn, loonberekening, inschrijving van nieuwe werknemers, contract wijzigingen, ziekte inbrengen en nog veel meer. De taken plannen gebeurt nu aan de hand van Windows Taakplanner maar deze applicatie is niet gebruiksvriendelijk. Ook geeft Windows Taakplanner geen enkele vorm van feedback dus weet de gebruiker niet of de taak geslaagd of gefaald is.

### Doelstelling

Tijdens deze stageopdracht wordt een applicatie ontwikkeld om Windows Taakplanner te vervangen. Deze applicatie moet gebruiksvriendelijk zijn zodat ze gebruikt kan worden door mensen zonder technische achtergrond. Ook is het de bedoeling dat de applicatie feedback toont zodat de gebruiker kan zien of en waarom een taak faalde. En als de taak geslaagd is, kan de gebruiker zien wat de taak allemaal gedaan heeft en hoelang hij duurde.

Ten slotte zijn er enkele beperkingen die het maken van een dergelijke tool bemoeilijken. Er wordt van uitgegaan dat de gebruiker de applicatie zal installeren op een gewone computer. Dit betekent dat er geen webserver ter beschikking is. Een ASP.NET-applicatie is dus niet mogelijk tenzij ze *self hosted* is en geen gebruikmaakt van Internet Information Service (IIS). Er kan ook geen databaseserver geïnstalleerd worden. Hiervoor moet een andere oplossing gezocht worden, zoals SQLite of tekstbestanden.

## Windows Taakplanner

Zoals vermeld is Windows Taakplanner niet gebruiksvriendelijk. In dit gedeelte van de paper wordt hier dieper op ingegaan.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Figuur 1 Windows Taakplanner

Wat opvalt aan de applicatie is dat ze er verouderd uitziet. De applicatie is relatief weinig veranderd sinds de versie die geïntroduceerd was in Windows Vista, dit is meer dan 14 jaar geleden.

Nog een probleem aan de taakplanner is dat er te veel informatie staat die niet nuttig is voor de gebruiker. Dit kan verbeterd worden door enkel de belangrijke informatie te tonen op het scherm. Als de gebruiker meer informatie wil over een taak kan hij deze selecteren en wordt er meer informatie getoond.

De gebruiker wordt niet echt begeleid door de applicatie. Dit is niet zo belangrijk voor gebruikers die veel ervaring hebben met computers maar voor iemand met minder technische kennis is dit heel ingewikkeld. Op het eerste zicht is het moeilijk om te vinden hoe een nieuwe taak moet aangemaakt worden. Dit kan best opgelost worden door wat extra contrast toe te voegen waarbij de belangrijke delen van de applicatie, zoals een nieuwe taak aanmaken, meer opvallen. Als de knop uiteindelijk wel gevonden is staan er twee mogelijkheden, namelijk ‘Basistaak maken’ en ‘Taak maken’. Hierbij is het niet direct duidelijk welke gekozen moet worden. Nadat de gebruiker wat verder zoekt komt hij erachter dat de tweede optie gewoon een meer uitgebreide aanpasbaarheid heeft.

# Uitwerking stageopdracht

## Informatie-uitwisseling

Er wordt dagelijks veel informatie uitgewisseld, zowel formeel als informeel. Voor formele communicatie gaat het vooral over team stand-ups en SCRUM-meetings. Deze gaan iedere dinsdag en donderdag door. Informele communicatie gebeurt via de Microsoft Teams-chat, zowel via tekstchat als videochat. Face-to-face-communicatie gaat dit jaar jammer genoeg niet omwille van Covid-19.

## Voorkomen van problemen

Om problemen te voorkomen worden er Unittesten geschreven.

### Unittesten

Unittesten zijn heel belangrijk voor het voorkomen van problemen. Een unittest controleert of de code doet wat hij moet doen. Het grootste voordeel van unittesten is dat ze veel sneller zijn dan handmatige testen. Dit betekent niet dat unittesten handmatige testen volledig vervangen. Unittesten testen namelijk enkel de werking van de code niet die van de effectieve applicatie. Een tweede voordeel van het schrijven van unittesten is kijken of bepaalde nieuwe aanpassingen de originele functie van de code niet verandert, de test zal falen als de code niet meer doet wat hij moet doen. Ten slotte is het ook handig voor regressie. Met regressie wordt bedoeld dat de testen falen wanneer de effectieve werking van de code verandert door een update van een gebruikt framework of een Windows-update.

### Code coverage

Code coverage geeft aan hoeveel van de totale code getest is. Dit zorgt voor een duidelijk overzicht van welke stukken code getest zijn en welke nog getest moeten worden. Hiervoor wordt gebruikgemaakt van de tool dotCover, een code coverage-tool die werkt binnen een .NET-omgeving en die werd ontwikkeld door JetBrains.

## Gebruikte technologieën

In dit deel van deze bachelorpaper worden alle gebruikte technologieën kort toegelicht.

### .NET Core

.NET Core is een gratis en opensourceframework gemaakt door voornamelijk Microsoft-ontwikkelaars. Het is de opvolger van .NET Framework. Het verschil tussen de twee frameworks is dat .NET Core cross-platform is. Dit betekent dat het zowel werkt op Windows, Linux als op MacOS. [2]

### Windows Presentation Foundation

WPF is een gratis en opensourcefrontendframework ontwikkeld door Microsoft. Het maakt gebruik van eXtensible Application Markup Language (XAML)-code om de frontend op te bouwen. WPF wordt ondersteund door alle Microsoft Windows-versies vanaf Windows Vista en Windows Server 2008. Maar op oudere versies van Windows kunnen de nodige bibliotheken geïnstalleerd worden om toch WPF te kunnen gebruiken.

WPF biedt veel features aan, zoals [3]:

* Direct3D: dit zorgt ervoor dat Windows een deel van de grafische berekeningen, die nodig zijn voor het renderen van de applicatie, op de Graphics Processing Unit (GPU) doet. Het gevolg hiervan is dat de Central Processing Unit (CPU) minder wordt belast.
* *Data binding*: hiermee kan de ontwikkelaar data koppelen aan frontendelementen. Dit is goed voor methodes zoals MVVM waarbij er geen referenties naar de view in de viewmodel mogen zijn. Er zijn vier verschillende versies hiervan.
  + One time binding
  + One way binding
  + Two way binding
  + One way to source binding
* Sjablonen: hiermee kan de stijl van de frontendelementen gemanipuleerd worden. Er zijn twee soorten:
  + Control templates
  + Data templates
* Animaties: WPF biedt veel steun aan voor het animeren van een applicatie. Hierbij gaat het zowel over kleine animaties aan de hand van timers als over meer ingewikkelde animaties aan de hand van animatieklassen.

### Entity Framework

Vroeger werd vaak gebruikgemaakt van ADO.NET-codeom informatie op te slaan of op te halen uit een database. Dit is een omslachtig en foutgevoelig proces. Daarvoor heeft Microsoft een framework ontwikkeld, namelijk Entity Framework.

Entity Framework is een open source Object-Relational Mapping (ORM)-framework gemaakt voor .NET-omgevingen. Het framework zorgt ervoor dat de ontwikkelaar zich kan focussen op het werken met objecten van domeinspecifieke klassen, zonder rekening te moeten houden met de databasetabellen en kolommen. Met Entity Framework kan de ontwikkelaar ook op een hoger niveau van abstractie werken. Dit is handig voor het toepassen van concepten als Dependancy Injection(DI). [4]

### Hangfire

Hangfire wordt in dit project gebruikt om taken automatisch op de achtergrond uit te voeren. Hangfire is één van de onderzoekstopic in dit eindwerk. In het onderzoek wordt hier dieper op ingegaan.

### Topshelf

Topshelf is een framework gemaakt voor het hosten van .NET-applicaties. Topshelf maakt het mogelijk om op een gemakkelijke manier een consoletoepassing te configureren als een Windows-service. Services zijn veel moeilijker om te debuggen dan consoletoepassingen, maar services zijn eenvoudiger om te installeren. Topshelf biedt hier het beste van beide werelden aan. [5]

### Serilog

Serilog is een logboekbibliotheek gemaakt om te gebruiken in .NET-omgevingen. Het is eenvoudig te configureren en biedt een gestructureerde Application Programming Interface (API) aan. Serilog is heel nuttig voor eenvoudige applicaties, maar het is eigenlijk gemaakt voor het loggen van complexe applicaties. Zo ondersteunt het bijvoorbeeld asynchrone toepassingen. [6]

### Prism

Prism is een framework dat de ontwikkelaar helpt bij het maken van XAML-applicaties die *loosely coupled*, onderhoudbaar en testbaar zijn. Het ondersteunt zowel WPF als Xamarin.Forms-applicaties. Prism implementeert een verzameling van ontwerppatronen die helpen bij het schrijven van goed gestructureerde code waaronder MVVM, DI, *commands*, *event aggregation* en nog vele anderen.

Het hoofddoel van Prism is om voor *cross-platform-*applicaties toch zo veel mogelijk gemeenschappelijke code te kunnen schrijven. Met Prism in Xamarin.Froms kan bijvoorbeeld een abstractie gebruikt worden voor navigatie die unittestbaar is. Daaronder kunnen dan de platform specifieke API’s aangesproken worden op een MVVM manier.

Prism is eenvoudig om te leren dankzij de uitgebreide documentatie op hun website en de vele videocursussen die online staan. Brain Lanugas en Dan Siegel bieden dit aan. Zij hebben beiden hun eigen YouTube-kanaal en Twitch-kanaal waarin zij video’s en livestreams delen over Prism. Hierbij gaat het zowel over tutorials als video’s over hoe dat zij bepaalde problemen oplossen. Ook is er een Pluralsight-cursus gemaakt door Brain Lanugas waar hij de basis van Prism toelicht. [7]

### NUnit

Unittesten zijn heel belangrijk tijdens het ontwikkelen van een applicatie. Een framework dat de ontwikkelaar hierbij kan helpen is NUnit. NUnit is een unittestframework gemaakt voor alle .NET- talen. Het is gebaseerd op JUnit, een unittestframework voor Java. [8]

NUnit biedt een aantal handige features aan, namelijk [9]:

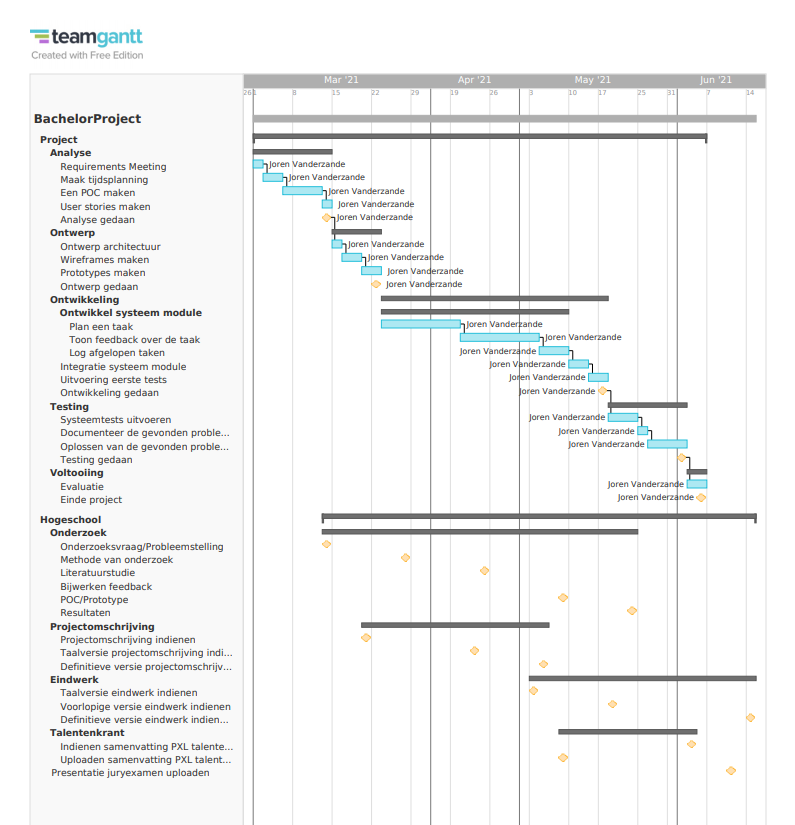
* De mogelijkheid om uitgevoerd te worden in consoletoepassingen, in Visual Studio via een testadapter en aan de hand van software van derden.
* De mogelijkheid om testen parallel uit te voeren.
* *Data* *driven* tests worden sterk ondersteund door NUnit.
* Ondersteuning voor meerdere platformen.

### Moq

Moq is de enige *mocking*-bibliotheek voor .NET die gebruikmaakt van Linq- en Lambda-expressies. Dit zorgt ervoor dat dit de meest productieve, *type-safe* en *refactor*-vriendelijke *mocking*-bibliotheek is binnen .NET. Een *mock* is eigenlijk een namaakobject dat het gedrag van het echte object simuleert. Moq ondersteunt zowel *mocking* van interfaces als klassen. De API van Moq is heel eenvoudig om te leren en de ontwikkelaar heeft geen voorkennis nodig over het *mocking*-concept. [10]

## Planning

Voor deze stageopdracht is er gekozen om gebruik te maken van een Gantt-grafiek. Een Gantt-grafiek is een soort van tijdlijn die aanduidt hoe het project zal verlopen. Het helpt om te bepalen hoelang het project zal duren en welke middelen er nodig zijn om het project tot een goed einde te brengen. Ten slotte is het handig om te zien of alles verloopt volgens schema. Dit zorgt ervoor dat het eerder duidelijk is of het project langer gaat duren dan verwacht en dat de planning moet worden aangepast. Voor deze planning is gebruikgemaakt van een gratis onlinetool, namelijk TeamGantt.



Figuur 2 Originele planning

Het project is opgedeeld in vijf hoofdtaken: analyse, ontwerp, ontwikkeling, testing en voltooiing.

De analysefase dient om duidelijk te maken wat de applicatie allemaal moet kunnen. Dit kan het best gebeuren door userstory’s te maken aan de hand van informatie verkregen uit de *requirements-meetings*. Ook moet er duidelijkheid zijn over welke frameworks gebruikt moeten worden. Hiervoor is een PoC gemaakt.

In de ontwerpfase is er een architectuurschema gemaakt. Hierin wordt duidelijk weergegeven welke frameworks en technologieën er gebruikt zullen worden in dit project en hoe ze verbonden zijn met elkaar. Ten slotte worden er wireframes en prototypes gemaakt zodat het duidelijk wordt hoe de applicatie eruitziet en hoe de applicatie werkt.

Voor de ontwikkelingsfase wordt de applicatie geprogrammeerd. Hierbij gaat het over frontend en backend. Ook worden er unittesten geschreven om de kwaliteit van de code hoog te houden.

In de testingfase gaat het vooral over handmatige testen om te kijken of de werking van de applicatie wel goed is.

Ten slotte worden er in de voltooiingsfase kleine aanpassingen gedaan aan de hand van feedback die gegeven wordt door de projectleider en de klant.

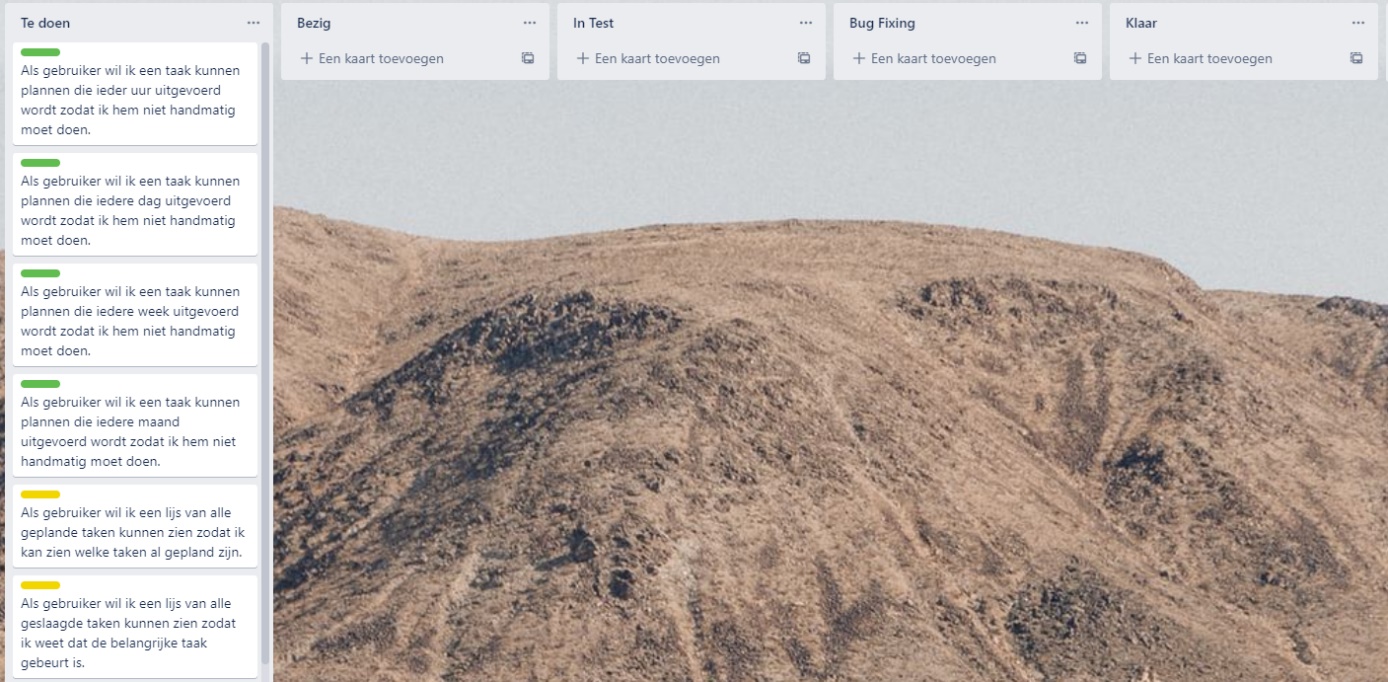
## Analyse

Aangezien nog niets zeker is binnen deze stageopdracht moet er grondig geanalyseerd worden wat de mogelijkheden zijn.

### Userstory’s

Userstory’s zijn korte beschrijvingen van wat de gebruiker van de applicatie allemaal wil kunnen en waarom de gebruiker dit wil. Dit geeft een duidelijk beeld over wat allemaal aanwezig moet zijn in de applicatie.

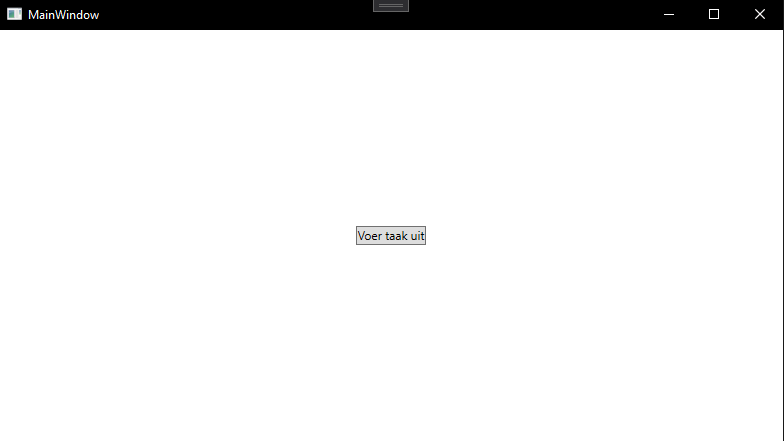
Normaal worden userstory’s geschreven door de *product owner* (de klant) maar omdat dit project op een agile manier zal verlopen worden ze geschreven door de ontwikkelaars. Een lijst van userstory’s heet de Product Backlog. Tijdens het ontwikkelproces worden er bepaalde userstory’s uit de Product Backlog gehaald en afgewerkt zodat er altijd een functioneel stuk wordt afgeleverd aan de klant. Om dit op een goede manier te doen is er een Kanbanbord gemaakt met Trello, een gratis onlinetool. [11]



Figuur 3 Kanbanbord

### Proof of Concept

Om uit te zoeken wat de mogelijkheden zijn en welke frameworks allemaal gebruikt gaan worden in het project, is het handig om een PoC te maken. Hiervoor is een WPF-applicatie gemaakt met een knop die een taak iedere minuut uitvoert. De RecurringJob klasse van Hangfire is hier een goede oplossing voor. In deze taak wordt er een simpel opdrachtprompt-commando uitgevoerd. Ook wordt er gebruikgemaakt van het Topshelf-framework om de Hangfire-server als Windows-service te installeren.



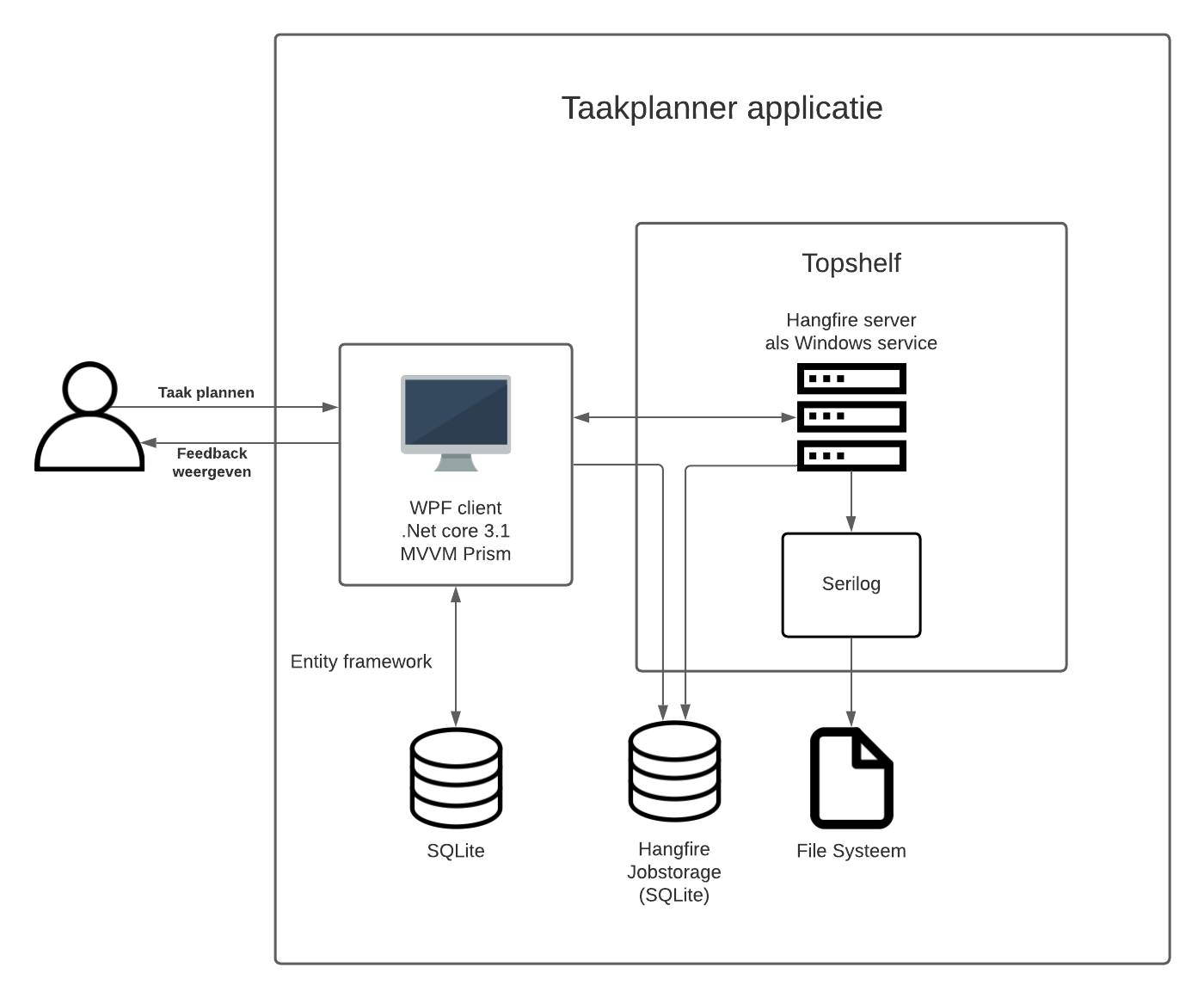
Figuur 4 Proof of Concept

Om uit te zoeken hoe het Hangfire-framework werkt, is er gebruikgemaakt van de Pluralsight-cursus “Simplefied .Net Background Tasks with Hangfire” door Rag Dhiman. In deze cursus legt hij uit hoe de Hangfire-server moet geconfigureerd worden en hoe taken kunnen aangemaakt worden in een ASP.NET-applicatie. Het aanmaken van een nieuwe taak in een WPF-applicatie gebeurt op dezelfde manier als in een ASP.NET-applicatie.

## Ontwerp

### Architectuurschema

Er is een architectuurschema gemaakt om te begrijpen hoe alle componenten van de applicatie met elkaar verbonden zijn. Dit geeft ook een goed overzicht van alle frameworks die gebruikt worden in deze applicatie. Het architectuurschema is gemaakt aan de hand van de gratis onlinetool Lucidchart.



Figuur 5 Architectuurschema

De gebruiker plant een taak in de WPF-client. Die taak wordt dan door Hangfire in de JobStorage toegevoegd. Aan de hand van Entity Framework wordt de extra data van de taak, die niets te maken heeft met Hangfire, opgeslagen, zoals een beschrijving van wat de taak precies doet. Hangfire heeft hier namelijk geen tabel voor in de database. Vervolgens kijkt de Hangfire-server, die als Windows-service is geconfigureerd, of er een taak is die moet worden uitgevoerd. Als hij een taak vindt, voert hij deze uit. Ten slotte schrijft Serilog de activiteit van de Hangfire-server naar een tekstbestand in het filesysteem.

### User experience design

Eén van de meest belangrijke criteria om deze stageopdracht tot een goed einde te brengen is gebruiksvriendelijkheid. Om de applicatie zo gebruiksvriendelijk mogelijk te maken is er onderzoek gedaan naar hoe dit best aangepakt wordt. Op het internet staan veel artikels en video’s die tips geven over hoe een applicatie opgebouwd moet zijn om een goede *user experience* te creëren. Djaya de Vries heeft één van deze artikels geschreven. Hij geeft vijf tips die helpen bij het maken van een gebruiksvriendelijke website. De meeste van deze tips kunnen ook overgenomen worden naar computerapplicaties. Hij gaf de volgende tips: [12]

#### Keep it Simple, Stupid

Dit is een uitspraak die vaak gebruikt wordt in de IT-wereld. Dit betekent dat het beter is om iets eenvoudigs te maken. Als het te ingewikkeld is, met allemaal toeters en bellen, wordt het eindresultaat onoverzichtelijk. Dit concept komt zowel terug bij het programmeren als bij het designen van een applicatie. Als een website/applicatie te veel inhoud bevat, weet de gebruiker niet direct waar hij moet klikken en waar hij zijn aandacht op moet focussen. Een eenvoudige website/applicatie is dus heel belangrijk, zeker aangezien de gemiddelde aandachtsspanne van een gebruiker maar acht seconden is. Soms is het dus beter om na te denken of er geen eenvoudigere manier is om een probleem op te lossen.

#### Omarm witruimte

Witruimte is de lege ruimte rondom de inhoud van een pagina, bijvoorbeeld de ruimte tussen grote delen van een pagina zoals tussen een *navbar* of een menubalk, maar ook de kleinere delen zoals ruimte tussen regels en knoppen. Dit zorgt ervoor dat de website/applicatie gemakkelijker is om te lezen en te scannen. Dit is niet het enige effect van witruimte. Witruimte zorgt ook voor een moderner uitzicht. Ten slotte helpt het ook om duidelijk te maken welke gedeeltes van de website/applicatie belangrijk zijn.

#### Optimaliseer de laadtijd van je website

Deze tip is minder toepasselijk voor computerapplicaties maar hier valt ook wel een les uit te leren. Het is namelijk heel belangrijk dat de performantie van de applicatie goed is. Als er dan toch een actie is die iets langer duurt, is het belangrijk dat de gebruiker een soort indicatie krijgt dat de applicatie weldegelijk iets bezig is, bijvoorbeeld een berekening die relatief moeilijk is afhankelijk van de computer of data halen uit een database met een slechte internetconnectie.

#### Maak de website interactief

Wat Djaya bedoelt met een interactieve website/applicatie is het gebruiken van micro-animaties, zoals een hover-effect, laadbalk of een scroll-animatie. Deze micro-animaties kunnen de gebruiksvriendelijkheid van een website/applicatie verbeteren ook al lijkt het heel simpel. Let op, dit soort animaties zijn heel leuk maar denk aan de eerste regel. Gebruik deze animaties niet te veel en zeker niet verkeerd.

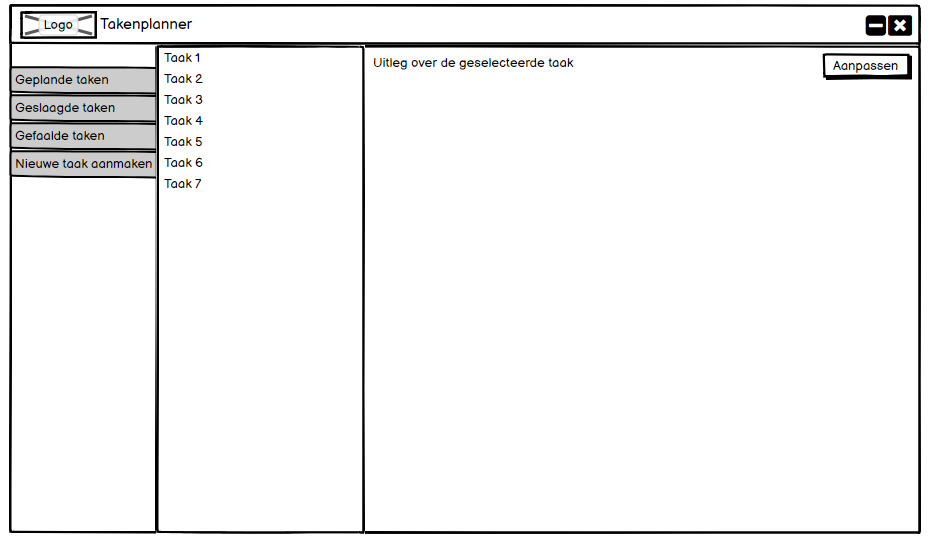
#### Maak de website responsive

Deze tip is weer minder toepasselijk voor een applicatie aangezien de applicatie niet gebruikt kan worden op een mobiel apparaat. Het is wel heel belangrijk dat de applicatie een beetje *responsive* is want verschillende gebruikers hebben verschillende schermresoluties of de gebruiker wil de applicatie misschien kleiner maken om twee applicaties tegelijk open te hebben.

### Wireframes en prototypes

Een wireframe is een visueel hulpmiddel bij het ontwikkelen van een website of applicatie. Het is een schets waarin een overzicht wordt gegeven van de verschillende onderdelen die aanwezig moeten zijn in de applicatie [13]. Prototypes daarentegen geven ook de werking van de applicatie weer. Ze hebben meestal kleur, animaties en de werkelijke inhoud die op de applicatie wordt weergegeven. Voor het maken van de wireframes en prototypes is Balsamic gebruikt, een gratis onlinetool.

#### Versie één wireframes

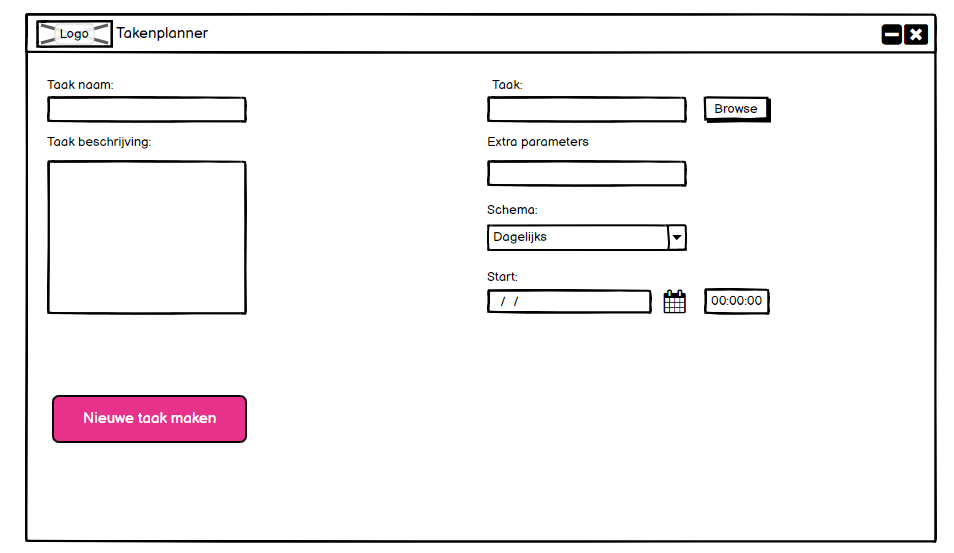


Figuur 6 Wireframe v\_1 Homepagina

De homepagina is opgedeeld in verschillende tabs:

* Geplande taken
* Geslaagde taken
* Gefaalde taken

Afhankelijk van welke tab geselecteerd is, toont de applicatie een aantal taken. De gebruiker kan op een taak drukken om meer informatie over de geselecteerde taak te verkrijgen. Ten slotte is er nog een “Nieuwe taak aanmaken”-knop die verwijst naar de “Nieuwe taak”-pagina.



Figuur 7 Wireframe v\_1 Nieuwe taak pagina

Op de “Nieuwe taak”-pagina verschijnen een aantal invulvelden:

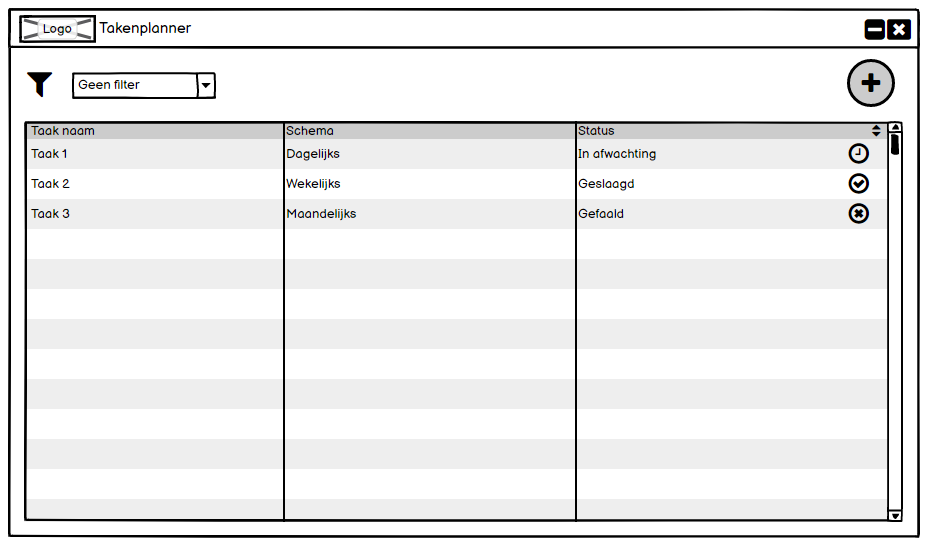
* Taak naam: in dit veld moet de gebruiker de naam van de taak invullen.
* Taak beschrijving: in dit veld moet de gebruiker een korte beschrijving van de taak invullen.
* Taak: in dit veld moet het pad naar de STA-file ingevuld worden, dit kan de gebruiker doen aan de hand van de browseknop.
* Extra parameters: in dit veld worden extra parameters toegevoegd.
* Schema: hier moet de gebruiker kiezen hoe vaak de taak uitgevoerd wordt.
* Start: hier moet de gebruiker de startdatum kiezen aan de hand van een *datepicker.*

Als de gebruiker alle velden heeft ingevuld kan hij op de “Nieuwe taak maken”-knop drukken om de taak op te slaan in de database. Deze knop is roze omdat dit één van de kleuren is in het logo van Acerta.

Deze wireframes waren niet zo goed. Vooral de homepagina kan verbeterd worden. De tab lay-out kan voor verwarring zorgen voor de gebruiker. De gebruiker moet een extra stap doorlopen om te weten op welke tab hij zit. Door alles op één pagina te zetten kan dit vermeden worden, maar dit brengt een aantal uitdagingen met zich mee. Alles op één pagina zetten, kan de pagina onoverzichtelijk maken.

De “Nieuwe taak”-pagina kan ook verbeterd worden. Er staat veel informatie op de pagina maar toch lijkt de pagina erg leeg. Er is niets mis met veel *whitespace* (lege ruimte) maar deze moet slim gebruikt worden en dus niet op deze manier.

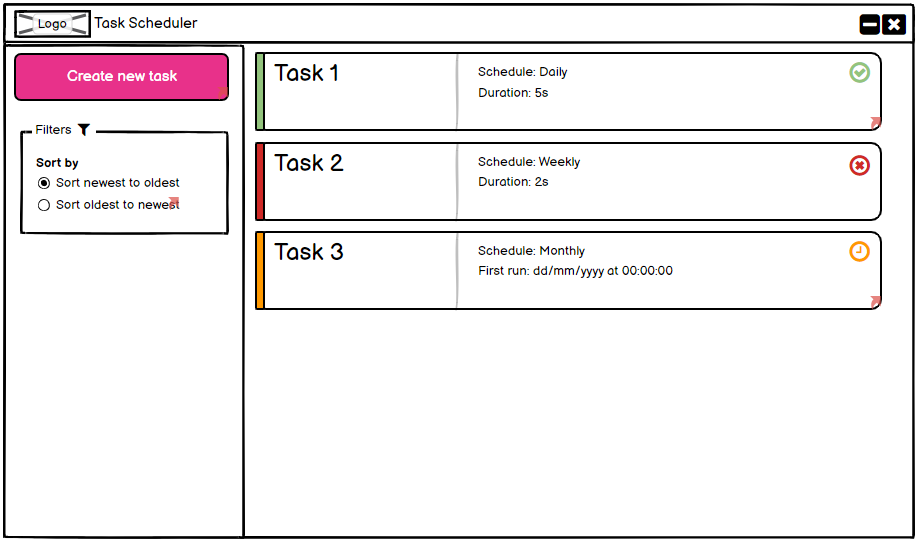
#### Versie twee wireframes



Figuur 8 Wireframe v\_2 Homepagina

In de tweede versie van de homepagina is gebruikgemaakt van een tabel met de belangrijkste informatie erin. Ook is er een embleem toegevoegd dat hoort bij de status van een taak. Dit lost het originele probleem met de tabs op maar dit heeft ook zijn nadelen. Het ziet er namelijk heel saai uit en is ook niet gemakkelijk om te stijlen. Om dit op te lossen kan best gekeken worden naar hoe mobile apps het tonen van veel data aanpakken. Smartphones hebben namelijk niet zoveel ruimte als een computerscherm.

#### Versie drie wireframes

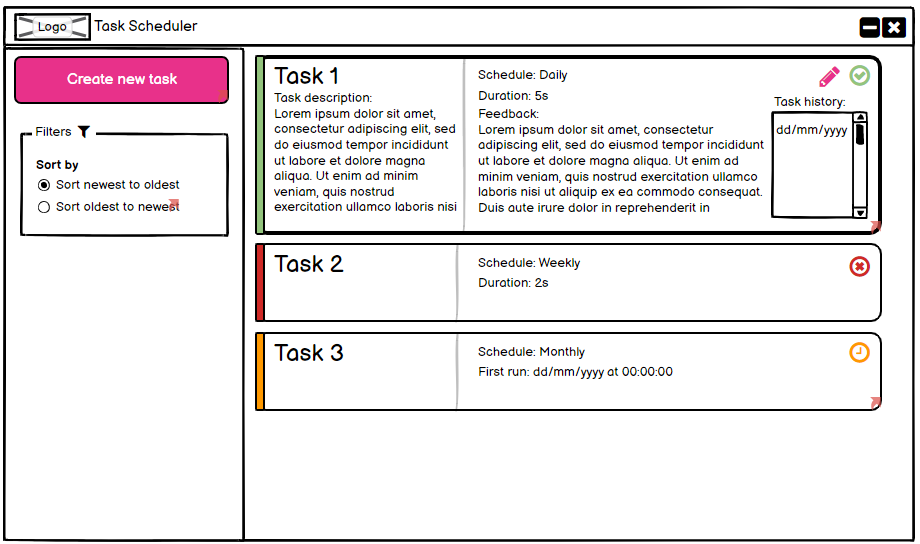


Figuur 9 Wireframe v\_3 Homepagina

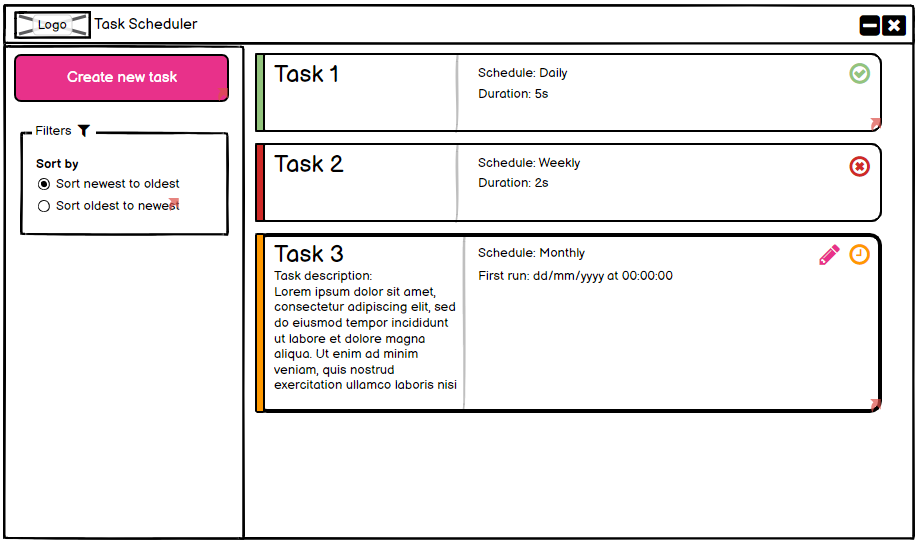
Voor de derde versie van de wireframes is er gevraagd om ze in het Engels te maken omdat Acerta een Belgisch bedrijf is en er meerdere talen gesproken worden. In deze versie van de wireframes is gekozen om de taken op te lijsten in een soort blokformaat. Hierin wordt de belangrijkste informatie van de taak getoond. De kleuren verwijzen naar de status van de laatst uitgevoerde taak:

* groen voor geslaagd
* rood voor gefaald
* oranje voor in afwachting.

Ook komen de emblemen van de vorige wireframes terug en bevindt zich aan de linkerkant van de pagina een menubalk met een knop en een filteroptie.

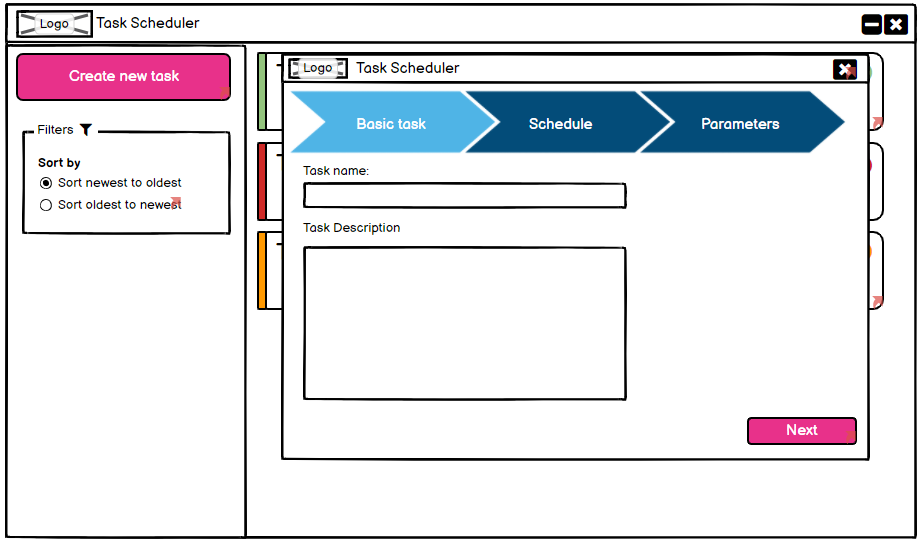


Figuur 10 Wireframe v\_3 Homepagina detail geslaagde taak

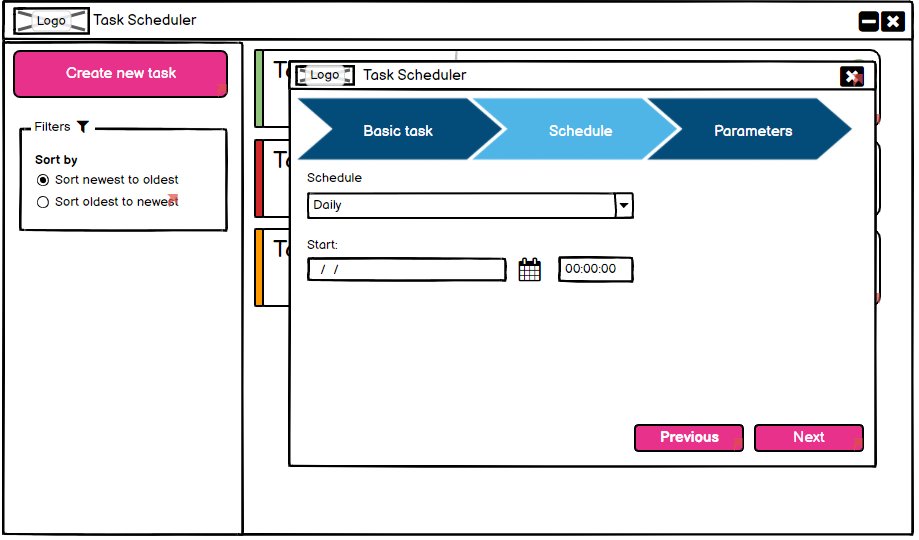


Figuur 11 Wireframe v\_3 Homepagina detail afwachtende taak

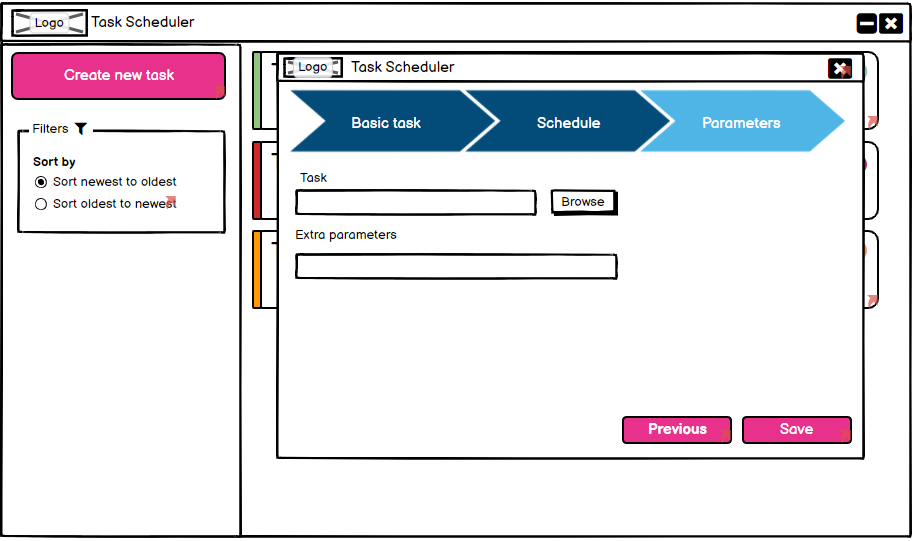
De geselecteerde taak wordt iets groter weergegeven. Er wordt meer informatie gegeven over de taak. Hierbij hoort bijvoorbeeld de beschrijving van de taak. Voor geslaagde en gefaalde taken wordt ook de feedback van de laatst uitgevoerde taak getoond, alsook een taakhistoriek zodat de gebruiker een taak uit het verleden kan bekijken.



Figuur 12 Wireframe v\_3 Nieuwe taak pagina basistaak



Figuur 13 Wireframe v\_3 Nieuwe taak pagina schema



Figuur 14 Wireframe v\_3 Nieuwe taak pagina parameters

Om *whitespace* te verminderen, maar vooral om de gebruiker meer te begeleiden bij het proces van het maken van een nieuwe taak, is er gekozen om gebruik te maken van een wizard. Er wordt nog steeds dezelfde informatie opgevraagd, maar deze keer is de pagina in drie verdeeld. De gebruiker kan tussen de verschillende pagina’s navigeren aan de hand van de knoppen in de hoek rechtsonder. Ook is er een indicatie, aan de hand van broodkruimels, die duidelijk maakt aan de gebruiker op welke pagina hij zich bevindt en hoeveel hij nog moet doen. Op de laatste pagina wordt de “*Next”-*knop een “*Save*”-knop om de taak op te slaan in de database.

## Ontwikkeling

### Projectstructuur

De *solution* is opgedeeld in 9 verschillende projecten. In dit gedeelte van het eindwerk worden deze projecten kort toegelicht.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Figuur 15 Projectstructuur

#### JobWizard

Prism biedt een *region*-systeem aan. Een *region* is een onderdeel in de applicatie waarvan de view veranderd kan worden. Dit zorgt voor een modulaire frontend. Voor iedere *region* is er een module gemaakt. De eerste module is de JobWizard-module. Hierin zitten alle views en *viewmodels* die nodig zijn voor het maken van een taak. Bij het navigeren naar een andere view wordt alle ingevulde informatie doorgegeven aan de view waarnaar genavigeerd wordt.

#### SchedulePicker

In de JobWizard-module zit nog een *region,* namelijk de SchedulePickerRegion. Hiervoor is ook weer een module gemaakt. In deze module zitten alle views en *viewmodels* die nodig zijn voor het kiezen van een schema waarop de taak wordt uitgevoerd. Ook wordt er in deze module de CronCalculator-klasse aangesproken om het ingegeven schema te converteren naar een cron-expressie.

#### TaskScheduler

Het TaskScheduler-project bevat de views en *viewmodels* van alle hoofdfrontendelementen, zoals de homepagina, de settingspagina en de JobWizard zelf. Zowel de settingspagina als de JobWizard zijn Prism *dialogs,* dit zijn pop-upvensters aangeboden door Prism.

#### TaskScheduler.ArnoTask

Dit project bevat alle logica om een commando uit te voeren in de opdrachtprompt. De enige klasse in dit project is de ArnoCommandTask-klasse. Deze klasse bevat een statische methode waarbij de ontwikkelaar alle parameters meegeeft die Arno nodig heeft om een taak uit te voeren. Dit is ook de methode die Hangfire zal uitvoeren op het ingegeven moment.

#### TaskScheduler.Core

In dit project bevindt zich alle gemeenschappelijk code. Dit project is later pas aangemaakt. Eerst was er maar één project, namelijk het TaskScheduler-project. Dit gaf problemen omdat de JobWizard-module en het TaskScheduler-project een referentie nodig hadden naar elkaar. Dit noemt een *circular dependency* en dit mag niet. In dit project zitten services, Data Transfer Objects (DTOs), Enums, Constante, enzovoort. Dit wordt allemaal gebruikt door meerdere projecten.

#### TaskScheduler.Data

In dit project zit de Entity Framework-code. Dit bestaat uit een datacontext waarin de databasestructuur wordt aangemaakt. Er zijn ook een aantal *repositories* die de data uit de database manipuleren. Ten slotte is er de *migrations*-folder. *Migrations* zijn files die Entity Framework nodig heeft om de database aan te maken. Deze worden gegenereerd door Entity Framework zelf, aan de hand van het Add-Migration-commando.

#### TaskScheduler.Domain

Dit project bevat de *domain*-klassen. Deze klassen worden zowel gebruikt in de frontend als in de database. De databasetabellen gemaakt door Entitiy Framework zijn namelijk gebaseerd op deze *domain*-klassen.

#### TaskScheduler.Hangfire

Dit is een consoleapplicatie waar de Hangfire-server wordt geconfigureerd. De Hangfire-server wordt geconfigureerd als een Windows-service zodat het gemakkelijk te installeren is. Ook is er een *Facades*-folder. In deze folder zitten twee klassen om taken aan te maken en op te vragen. Het is niet nodig om een aparte service hiervoor te maken omdat deze methodes al voorzien worden in Hangfire zelf. Dit is toch gedaan om de code testbaar te houden. De functionaliteit van Hangfire moet niet getest worden maar er moet wel gecontroleerd worden of de methode wel wordt uitgevoerd.

#### TaskScheduler.Test

Dit project bevat alle unittesten voor de applicatie. De testklassen zijn opgedeeld per *namespace*. Er wordt gestreefd naar een hoge *code coverage*.

### Integratie

De taakplanner moet op zo’n eenvoudig mogelijke manier geïntegreerd worden in Arno, de reeds bestaande software van Acerta. Met zo eenvoudig mogelijk wordt bedoeld dat de integratie automatisch kan gebeuren en dat er zo weinig mogelijk files nodig zijn om de applicatie te kunnen gebruiken. Om dit te doen kan een .exe-file gemaakt worden die alle *dependencies* bevat. De .exe-file is dan redelijk groot maar als de gebruiker deze file op zijn apparaat heeft, kan hij de applicatie gewoon gebruiken zonder enige installatie.

.NET heeft hier een commando voor, namelijk “dotnet publish”. Dit commando genereert alle files die nodig zijn om de applicatie te kunnen gebruiken. Er zijn wel een aantal parameters nodig die ervoor zorgen dat alles in één .exe-file zit.

Eerst moet er bepaald worden welke *build*-configuratie gegenereerd moet worden. Er zijn twee verschillende versies hiervoor, namelijk “debug*”* en “release*”*. Omdat het over integratie gaat moet er gekozen worden voor “*release”*. De parameter die hiervoor kan toegevoegd worden is “-c release”. Als deze parameter niet meegegeven wordt, staat de *build*-configuratie standaard op “debug*”*.

De tweede parameter die moet worden toegevoegd is “--self-contained”. Deze parameter zorgt ervoor dat de .NET runtime samen met de applicatie gepubliceerd wordt, zodat de runtime niet meer geïnstalleerd moet worden op het apparaat van de gebruiker.

Voor de tweede parameter is er nog een extra parameter nodig. Er moet namelijk een runtime-environment gekozen worden. Hiermee wordt bedoeld op welk soort apparaat wordt de applicatie geïnstalleerd. In dit geval is er gekozen voor 64-bit Windows-systemen. Dit kan gedaan worden door “-r win-x64” toe te voegen aan het commando. Een aantal andere voorbeelden van deze runtime-environments zijn osx-x64 (64-bit Mac-systemen), linux-x64 (64-bit Linux-systemen) en win-x86 (32-bit Windows-systemen).

Om de applicatie ten slotte te verpakken in een .exe-file, wordt “-p:PublishSingleFile=true” toegevoegd aan het commando. Wanneer de applicatie voor de eerste keer wordt uitgevoerd, worden alle nodige files uitgepakt naar een folder op basis van de applicatienaam en de *build identifier. D*it kan even duren. De volgende keren dat de applicatie wordt opgestart, is dit niet meer nodig en zal dit dus veel sneller gaan.

Het hele commando, met alle parameters samen, ziet er dus zo uit: “dotnet publish -c release --self-contained -r win-x64 -p: PublishSingleFile=true”. Dit genereert een .exe-file in de bin-folder van het project. Het commando moet uitgevoerd worden voor ieder uitvoerbaar project, in het geval van deze applicatie zijn dat er twee, namelijk het Taskscheduler- en TaskScheduler.Hangfire-project.

### Eindresultaat

#### Homepagina

De homepagina heeft rechtsonder aan het scherm een plusknop. Als de gebruiker met zijn muis over de plusknop gaat, komen er twee opties tevoorschijn, namelijk “Creat new task” en “Settings”.

Afbeelding met tekst, whiteboard

Automatisch gegenereerde beschrijving

Figuur 16 Homepagina resultaat

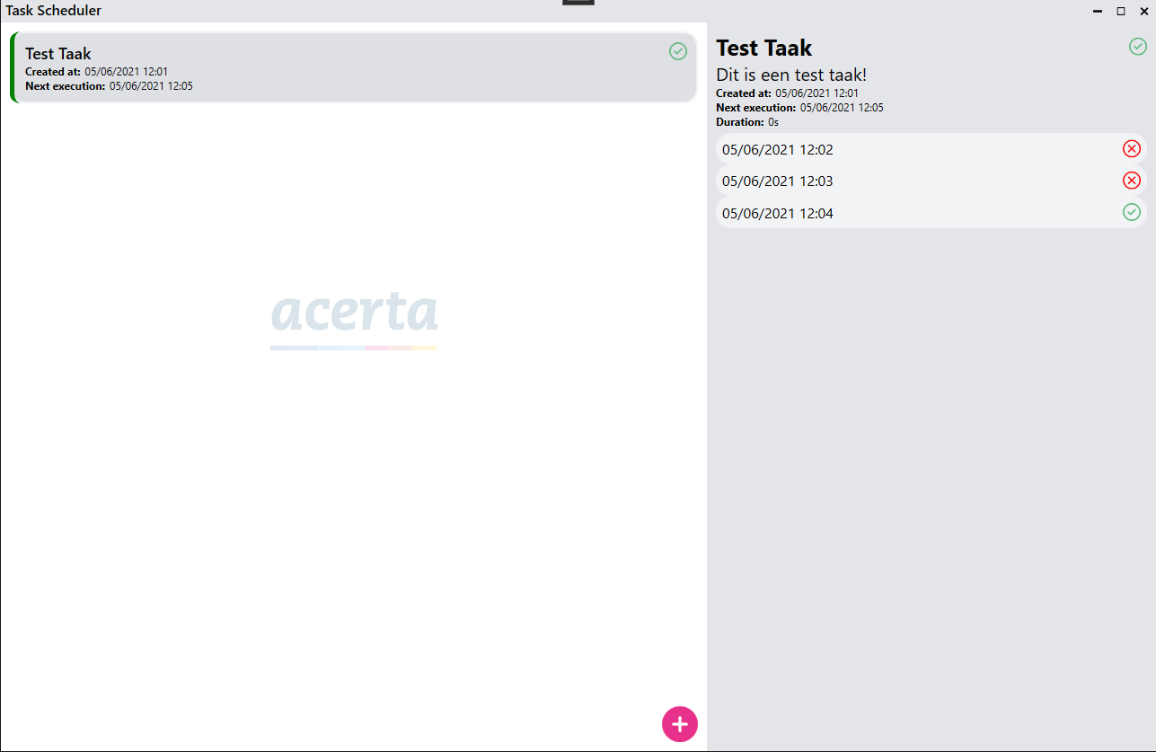
Standaard staat er alleen de meest belangrijke informatie op het scherm:

* de naam van de taak
* de datum wanneer de taak is aangemaakt
* de datum waarop de taak de volgende keer wordt uitgevoerd
* een symbooltje dat aanduidt of de taak geslaagd, gefaald of nog niet uitgevoerd is. Afhankelijk van deze status krijgt de taak ook een kleur: groen voor geslaagd, oranje voor in afwachting en rood voor gefaald.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Figuur 17 Homepagina met taak resultaat

Als de gebruiker op een taak klikt, wordt er meer informatie getoond over de geselecteerde taak. Hierbij schuift er een scherm open waarop dezelfde informatie weer te zien is. Er staat ook nog extra informatie, namelijk de looptijd van de laatst uitgevoerde taak en een historiek van iedere keer dat deze taak uitgevoerd is.

Figuur 18 Homepagina detail resultaat

#### Settings

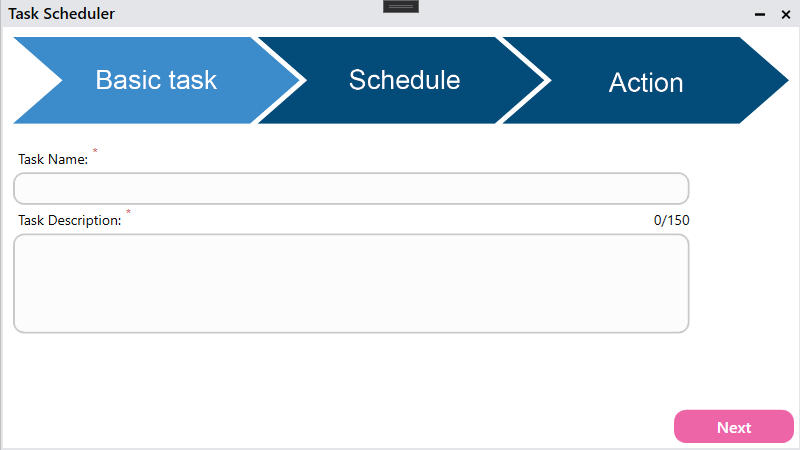
Om taken uit te kunnen voeren moet de gebruiker ingelogd zijn op Arno. Hiervoor kan een gebruikersnaam en wachtwoord meegegeven worden in het settingsscherm. Ook moet de taak uitgevoerd worden in een bepaald dossier. De gebruiker kan zoveel dossiers toevoegen als hij wil.

#### Afbeelding met tekst Automatisch gegenereerde beschrijving

Figuur 19 Settingspagina resultaat

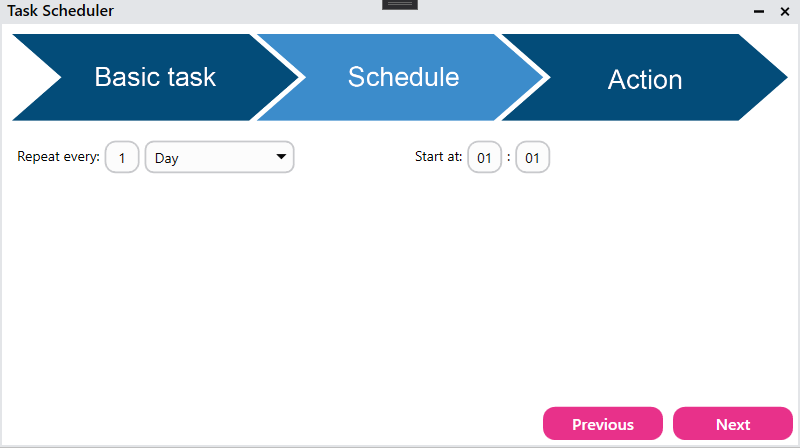
#### Nieuwe taak wizard

Om een taak aan te maken is er een wizard voorzien die de gebruiker begeleidt door het proces om een nieuwe taak te maken. De eerste pagina van de wizard is de basistaakpagina. Op deze pagina moet de gebruiker de naam en de beschrijving van de taak meegeven. De beschrijving mag maximum 150 karakters lang zijn. Hiervoor is een teller voorzien zodat de gebruiker altijd weet hoeveel hij nog mag schrijven. Zowel de naam als de beschrijving van een taak zijn verplicht om in te vullen, de “Next”-knop is niet actief totdat deze velden zijn ingevuld.

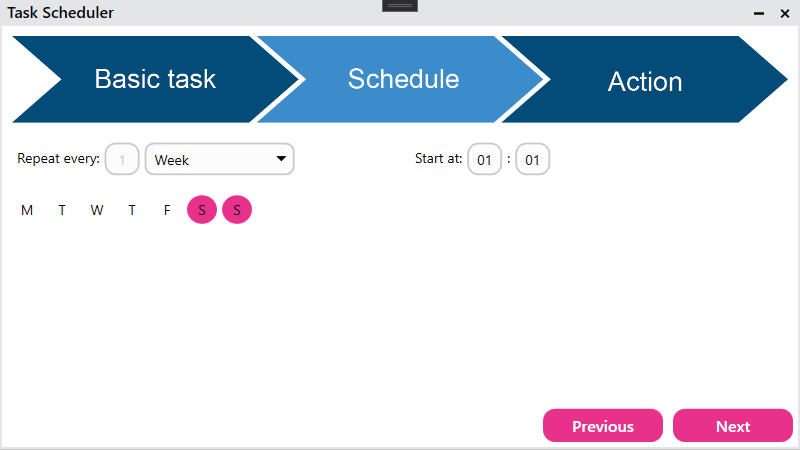


Figuur 20 Job wizard basis taak resultaat

Op de tweede pagina van de wizard kan de gebruiker het schema waarop de taak wordt uitgevoerd invullen. De gebruiker kan kiezen tussen de volgende schema’s: dagelijks, wekelijks, maandelijks, jaarlijks en *continuation*. Het dagelijkse schema is redelijk simpel, de gebruiker moet enkel meegeven om de hoeveel dagen de taak uitgevoerd moet worden. Er moet ook meegeven worden om hoe laat de taak wordt uitgevoerd maar dit moet bij ieder schema behalve bij het *continuation*-schema.



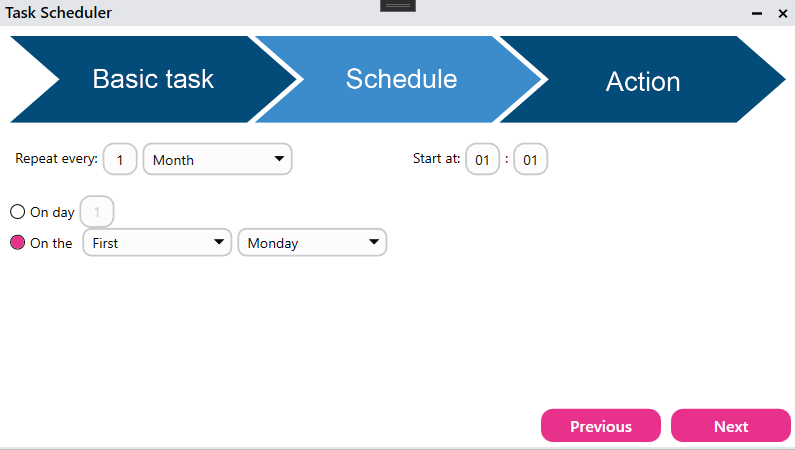
Figuur 21 Job wizard dagelijks schema resultaat

Bij het wekelijkse schema wordt meegegeven op welke dagen van de week de taak uitgevoerd moet worden. 

Figuur 22 Job wizard wekelijks schema resultaat

Op het maandelijkse schema kan de gebruiker kiezen tussen twee opties:

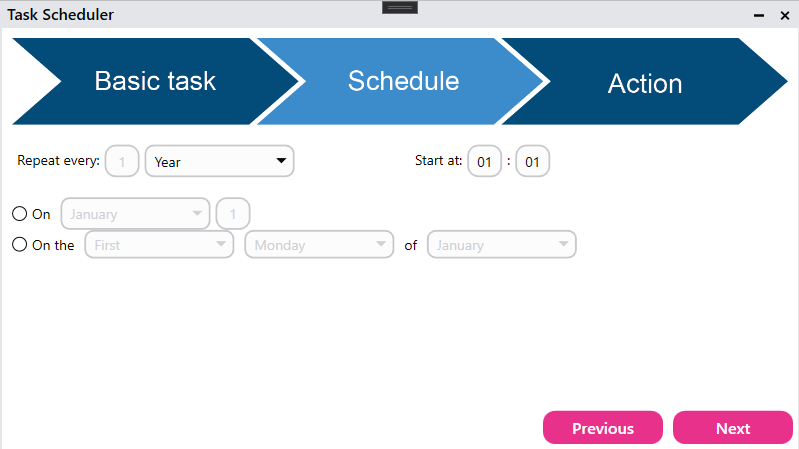
* een vaste datum van de maand (bijvoorbeeld de 22ste van de maand)
* op een vaste dag van de maand (bijvoorbeeld de eerste maandag van de maand).



Figuur 23 Job wizard maandelijks schema resultaat

Voor een jaarlijks schema zijn er ook twee opties:

* elk jaar op een vaste datum van een bepaalde maand (bijvoorbeeld ieder jaar op 31 januari)
* elk jaar op een vaste dag van een bepaalde maand (bijvoorbeeld ieder jaar op de eerste maandag van januari).



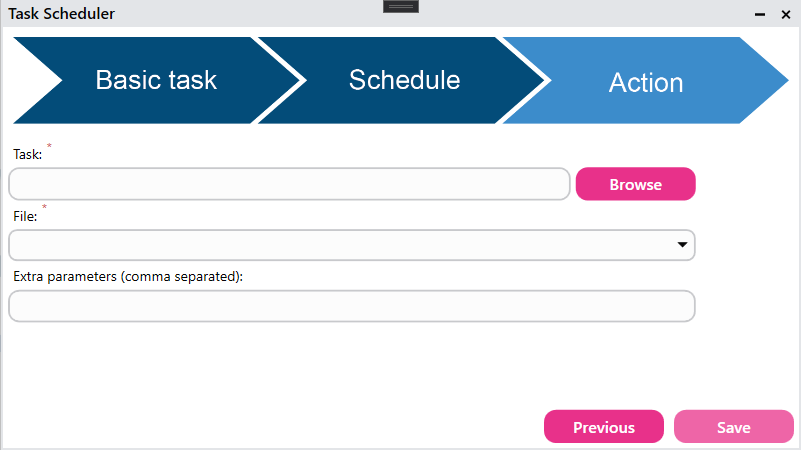
Figuur 24 Job wizard jaarlijks schema resultaat

Ten slotte is er het *continuation*-schema, hier kan de gebruiker kiezen om een taak uit te voeren vlak na een andere taak. Het tijdstip van de taak moet dus niet worden meegegeven.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Figuur 25 Job wizard continuation-schema resultaat

Op de laatste pagina van de wizard wordt de actie en zijn parameters meegegeven. Bij het taakveld kan de gebruiker, aan de hand van de browseknop, een STA-file selecteren. De filedropdown is gevuld met de dossiers van het settingsscherm. En ten slotte is er ook de mogelijkheid om extra parameters toe te voegen aan de taak. Zowel de taak als het dossier zijn verplichte velden.

Figuur 26 Job wizard actie resultaat

## Testing

### Façadepatroon

Een façadeklasse is een klasse die de onderliggende complexe code van een andere klasse maskeert om het beter leesbaar en bruikbaar te maken voor de ontwikkelaar. Maar dit is niet het enige waarvoor façadeklassen gebruikt kunnen worden. Binnen heel het testingverhaal van een applicatie kan een façadeklasse gebruikt worden om statische klassen of klassen waarover de ontwikkelaar geen controle heeft (bijvoorbeeld klassen uit een andere bibliotheek) te *mocken*. De mock kan dan gebruikt worden om te controleren of een bepaalde methode van die klasse opgeroepen wordt. Hij kan ook gebruikt worden om te bepalen wat de methode terugstuurt en ten slotte kan hij ook de methode een exceptie laten gooien. Dit designpatroon is gebruikt in deze stageopdracht om bepaalde methodes te testen die bijvoorbeeld gebruikmaken van Hangfire- of Prism-functionaliteit.

### Code coverage

Zoals al vaker vermeld in dit eindwerk, zijn unittesten heel belangrijk. Daar is dus tijdens deze stage ook veel tijd ingekropen. Er is gestreefd naar een zo’n hoog mogelijke code coverage. De code coverage van dit project is 86%. De resterende 14% zijn methodes en klassen die geen ingewikkelde zelfgemaakte bedrijfslogica bevatten. Hieronder vallen de façadeklassen, de views, Het Taskscheduler.Hangfire-project en het TaskScheduler.ArnoTask-project.Afbeelding met tekst, scorebord, schermafbeelding

Automatisch gegenereerde beschrijving

Figuur Test coverage rapport

# Extra stageopdracht: Talkdesk Proof of Concept

Omdat de eerste stageopdracht goed vooruitging, is er gevraagd om nog een extra project op te nemen. Origineel was de planning om dit samen met één andere stagiair te doen maar door een miscommunicatie in het team is deze stageopdracht uiteindelijk door vier stagiairs gedaan.

## Wat is Talkdesk

Talkdesk biedt een cloudbasedoplossing aan voor het managen van callcenters. De software heeft een eenvoudige interface die geavanceerde functionaliteit, uitgebreide rapportage en integratie met meer dan vijfentwintig bedrijfsprogramma’s biedt aan hun gebruikers. [14] Ook heeft Talkdesk een API die aangesproken kan worden vanaf een zelfgemaakte applicatie. De API is gebaseerd op het RESTful-principe en is beveiligd met Oauth 2.0-verificatie. [15]

## Doelstelling

Het doel van deze opdracht is om uit te zoeken wat de mogelijkheden zijn met de Talkdesk API.

* Welke data kan eruit gehaald worden?
* Is de data die eruit gehaald wordt live?
* Kan de data opgehaald worden door een zelfgemaakte applicatie?

Dit zijn allemaal vragen die beantwoord moeten worden aan de hand van deze PoC.

## Uitwerking extra opdracht

### Authenticatie

Om de API aan te kunnen spreken moet er een JSON Web Token (JWT) meegegeven worden aan iedere *request*. Om deze JWT-token op te kunnen vragen moet er een *request* gestuurd worden naar de token *end-point.* Dit gebeurt aan de hand van Basic Authentication en een Base64 *encoding* van de client­\_id en de client\_secret.

### Reporting API

Als de JWT-token is opgevraagd, kan er gebruikgemaakt worden van de *reporting* API van Talkdesk. Om de *reporting* API te gebruiken, moet er een rapport aangemaakt worden. Dit kan gedaan worden door een tijdspanne en de JWT-token mee te geven. Vanaf het moment dat het rapport aangemaakt is, kan het opgevraagd worden. Hiervoor is de *identifier* van het aangemaakte rapport nodig. Deze kan uit de vorige *request* gehaald worden.

De data die uit de *reporting* API gehaald kan worden:

* totaal aantal oproepen
* totaal aantal inkomende oproepen
* totaal aantal uitgaande oproepen
* totaal aantal oproepen die op tijd waren
* totaal aantal oproepen die te laat waren
* gemiddelde wachttijd van de oproepen

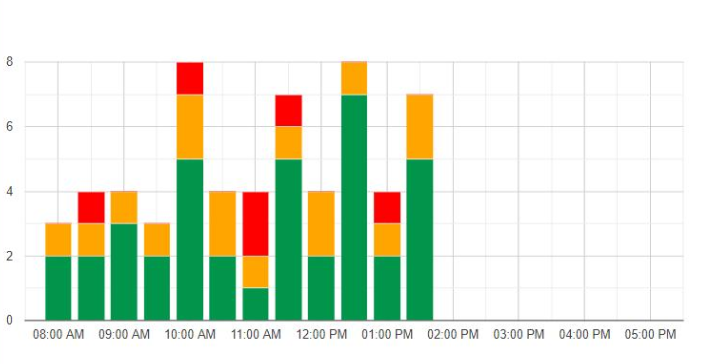
### Backend

De Talkdesk API kan niet rechtstreeks aangesproken worden vanuit de browser. Indien de ontwikkelaar dit probeert, stuurt de API een Cross-Origin Resource Sharing (CORS)-error terug. Om dit op te lossen is een Rest API gemaakt met ASP.NET die de data ophaalt wanneer de *end-points* van deze API aangesproken worden.

### Frontend

Voor de frontend is er gebruikgemaakt van HyperText Markup Language (HTML), Cascading Style Sheets (CSS) en JavaScript. In JavaScript wordt aan de hand van de Fetch API de backend aangesproken. De data, die via de backend doorgegeven wordt, wordt dan verwerkt en getoond op het scherm met HTML.

De frontend toont een grafiek die de activiteit van de hele dag weergeeft. Groen betekent dat de oproep op tijd was, oranje betekent dat de oproep bijna te laat was en rood betekent dat de oproep te laat was.



Figuur 28 Talkdesk voorbeeld grafiek

### Conclusie

Uit deze PoC blijkt dat de mogelijkheden met Talkdesk wel redelijk groot zijn. Alle data die de klant belangrijk vindt, kan opgehaald worden. Er is echter één groot nadeel met de API, namelijk de data die opgehaald wordt, kan niet live worden opgehaald. Er is altijd één uur vertraging op de data.

1. Onderzoekstopic

# Vraagstelling onderzoek

Het handmatig uitvoeren van achtergrondtaken kost veel tijd en energie en is foutgevoelig, zeker als het over repetitieve taken gaat. Er bestaan heel wat tools die dit proces kunnen automatiseren.

Voor dit project zijn er bepaalde beperkingen die het maken van zo’n tool bemoeilijken. Er wordt van uitgegaan dat de gebruiker de applicatie op een gewone computer wil installeren. Dit betekent dat de gebruiker geen webserver of databaseserver heeft. Ook is het de bedoeling dat deze applicatie opdrachtprompt-commando's uitvoert.

Voor dit project is er gekozen om gebruik te maken van Hangfire, een .NET-framework, omdat er bij Tobania al kennis was over dit framework. Dit is niet de enige tool die hiervoor gebruikt kan worden. Azure Logic Apps, een tool waarmee workflows kunnen geautomatiseerd worden zonder veel code te moeten schrijven, is hier een goed voorbeeld van. [16] Ook is er binnen de .NET-omgeving nog een ander framework dat voor automatisering zorgt, namelijk Quartz.NET.

Er zal onderzoek gedaan worden naar deze tools om uit te zoeken of Hangfire wel de juiste oplossing is. Zo niet? Welke is dan beter? En Waarom? Dit gebeurt aan de hand van een aantal ondervragen, namelijk:

* Kan de tool gebruikt worden zonder webserver?
* Kan de tool gebruikt worden zonder databaseserver?
* Kan de tool opdrachtprompt-commando’s uitvoeren?
* Hoeveel kost de tool om te gebruiken?
* Hoe gemakkelijk/moeilijk is het om de tool te leren gebruiken?

# Onderzoeksmethode

Het doel van de stageopdracht is dat er binnen een bestaande omgeving taken automatisch opgestart kunnen worden. Hiervoor is er gebruikgemaakt van een Hangfire-server die als Windows-service ingesteld is. Het opslaan van de taken gebeurt in een SQLite-database, om de applicatie *lightweight* te houden. Ten slotte is de frontend gemaakt in WPF.

Om te bepalen welke technologie het beste is voor deze toepassing zal er een lijst worden samengesteld van de voor- en nadelen van de gekozen technologieën. Dit zal proefondervindelijk gebeuren aan de hand van een PoC. Hiervoor zal er een applicatie worden gemaakt. Op deze applicatie is er een dropdownmenu met de volgende keuzes: iedere minuut, dagelijks, wekelijks. Daarna zal de applicatie op het gevraagde tijdstip een ping-commando uitvoeren.

Het ontwikkelproces van deze applicaties wordt dan vergeleken met elkaar aan de hand van diverse criteria. De applicatie moet standalone zijn. Hiermee wordt bedoeld dat er geen webserver gebruikt kan worden zoals IIS Express. Omdat het niet zeker is dat de gebruiker een database heeft op zijn computer, moet dit opgelost worden op een andere manier, zoals aan de hand van een SQLite-database of tekstbestanden. Ten slotte moet de applicatie opdrachtprompt-commando's kunnen uitvoeren.

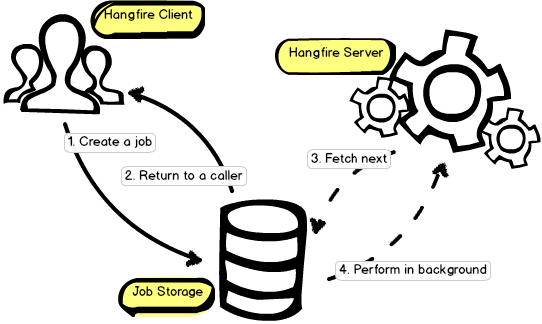
# Technologieën

In dit hoofdstuk van het onderzoek worden de tools en frameworks, die gekozen zijn, toegelicht.

## Hangfire

Hangfire is een opensourceframework. Het framework is gemaakt om binnen een .NET-omgeving automatisch taken te kunnen uitvoeren. Het is heel gemakkelijk om achtergrondtaken te plannen. [17] Hangfire heeft geen Windows-service of apart proces nodig om te kunnen werken. [18] Hangfire is volledig gratis, zelfs voor commercieel gebruik. Er zijn een aantal abonnementen die toegang geven tot extra features. Hiervoor moet jaarlijks een bepaald bedrag betaald worden. [17]

Hangfire werkt aan de hand van drie verschillende componenten, namelijk de client, het opslaggeheugen en de server.



Figuur 29 Simpel architectuurschema [19]

Op bovenstaande afbeelding voegt de gebruiker een taak toe aan de database. De Hangfire-server haalt de taak uit de database op en voert deze uit op de achtergrond. [20]

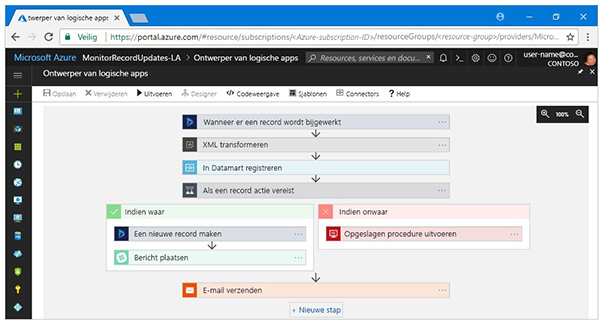
## Quartz.NET

Quartz.NET is een opensourceframework. Het framework is gemaakt om binnen een .NET-omgeving automatisch taken te kunnen plannen. Quartz.NET is geporteerd vanuit een populair Java framework. [21] Aangezien het een opensourceframework is, is het volledig gratis om te gebruiken voor commercieel gebruik. Quartz.NET kan gebruikt worden voor kleinschalige systemen maar ook voor grote ondernemingssystemen. [22]

Quartz.NET is volledig uitgerust met de benodigde tools om automatisch taken uit te voeren op een goede manier. Dit betekent dat er niet allerlei extensies nodig zijn om een applicatie te bouwen. Het framework biedt uitgebreide steun aan voor het werken met cron-expressies. [22] Een cron-expressie is een commando dat repetitieve taken op een bepaald tijdstip uitvoert. [23]

## Azure Logic Apps

Microsoft Azure Logic Apps is een cloudbasedervice waarmee ontwikkeltaken kunnen uitgevoerd worden. De ontwikkelaar kan hiermee op een gemakkelijke en begrijpelijke manier zijn workflows automatiseren. Veel code is hier niet voor nodig. [16] Met de visuele designer kan een reeks van stappen automatisch uitgevoerd worden op basis van triggers en acties.



Figuur 30 Voorbeeld visuele designer [16]

De workflow start met een trigger. Deze geeft aan de app door wanneer de workflow mag gestart worden. Als de workflow is geactiveerd worden er bepaalde acties uitgevoerd. Elke actie verwijst naar een bewerking die wordt gedefinieerd door een connector. Deze maakt een verbinding met *third*-party-software. [16]

# Literatuurstudie

## Hangfire

### Dashboard

Volgens Jimmy De Ruysscher, een softwareontwikkelaar bij Foreach met meer dan zestien jaar ervaring, is Hangfire een mooi, licht, makkelijk te gebruiken framework. Het geeft een mooi dashboard weer waarin bepaalde taken bekeken kunnen worden en indien nodig kunnen worden aangepast. [24] Dit wordt ondersteund door Joydip Kanjilal, een Microsoft Most Valuable Professional (MVP) in ASP.NET. Hij zegt: “The documentation for Hangfire is very detailed and well structured, and the best feature is its built-in dashboard. The Hangfire dashboard shows detailed information on jobs, queues, status of jobs, and so on.” [18].

### Soorten taken

Er is een verscheidenheid aan taken die uitgevoerd kunnen worden met Hangfire, namelijk Fire-and-forget Jobs, Delayed Jobs, Recurring Jobs en Continuations. Mukesh Murugan, een .NET (C#)-professional met meer dan vier jaar ervaring in softwareontwikkeling, legt deze taken verder uit aan de hand van een voorbeeld en komt tot volgende toelichting [17]:

* Fire-and-forget Jobs zijn taken die slechts één keer en onmiddellijk worden uitgevoerd. Het bedrijf stuurt eenmalig een e-mail naar een klant.
* Delayed Jobs of uitgestelde taken: in plaats van de e-mail onmiddellijk te versturen wordt deze pas na 10 minuten verstuurd.
* Recurring Jobs of terugkerende taken: in dit voorbeeld neemt de klant een abonnement en moet er elke maand een e-mail worden verstuurd naar de klant. Dit gebeurt door middel van een cron-expressie.
* Continuations of vooruitzettingen van andere taken: dit is een ingewikkelder scenario. Hierbij moet een taak pas worden uitgevoerd vanaf het moment dat een voorgaande taak voltooid is. In het voorbeeld wil de klant het abonnement opzeggen. Eerst moet de klant uitgeschreven worden en daarna mag er pas een e-mail verstuurd worden ter bevestiging van de uitschrijving.

### Documentatie

Op de website van Hangfire, Hangfire.io, staat allerlei informatie. Een van de dingen die zij aanbieden is een documentatie van hoe het framework werkt. Joydip Kanjilal vindt de documentatie heel uitgebreid en gestructureerd. [18] Ilya Chumakov, lead C# ontwikkelaar bij bi.zone, zegt zelfs “In addition, the quality of the documentation is perfect.” [20]

### Windows-service

Ilya Chumakov vermeldt ook dat Hangfire gebruikt kan worden in veel soorten applicaties. Zo kan het bijvoorbeeld gebruikt worden in een Windows-service, consoleapplicatie of een webapplicatie aan de hand van ASP.NET. [20] Dit wordt ook aangehaald in de bron van Joydip Kanjilal. [18] In deze stageopdracht is Hangfire wel geconfigureerd als Windows-service om de installatie zo eenvoudig mogelijk te houden.

### Opslagmogelijkheden

Volgens Jimmy De Ruysscher zijn er veel opslagmogelijkheden die ondersteund worden door Hangfire. De taken worden opgeslagen in een opslagmogelijkheid naar keuze. Het voordeel hieraan is dat als de applicatie afgesloten wordt, willekeurig of door de gebruiker, de taken nog altijd veilig zijn opgeslagen. Vanaf het moment dat de applicatie terug opgestart wordt, staan de taken klaar om uitgevoerd te worden. [24]

### Prijs

In de bron van Mukesh Murugan en die van Ilya Chumakov wordt vermeld dat Hangfire gratis is. Dit komt omdat het een opensourceframework is. Er zijn echter een aantal abonnementen waarvoor betaald moet worden. Deze geven de ontwikkelaar toegang tot extra features. De belangrijkste features zitten wel in het gratis pakket van Hangfire. [17] [20]

### NuGet

Hangfire biedt ook veel extensies aan. Deze worden gemaakt door ontwikkelaars die geïnteresseerd zijn in het Hangfire-project, maar die vinden dat er bepaalde features ontbreken. Deze extensies kunnen gemakkelijk geïnstalleerd worden aan de hand van NuGet. Dit zorgt ervoor dat de mogelijkheden van Hangfire heel breed zijn. Hoewel dit een groot voordeel is, heeft het ook nadelen volgens Thomas Hansen, Head Of Development bij TTCM Traders Trust. Hangfire is iets te uitgebreid voor bepaalde scenario’s. [21]

## Quartz.NET

### Taken, triggers en planners

Joydip Kanjilal zegt dat er drie belangrijke concepten zijn binnen Quartz.NET, namelijk taken, triggers en planners. Een taak is het stuk code dat moet uitgevoerd worden op het juiste moment. Een taak moet overerven van de IJob-interface. Dit is een bestaande interface in Quartz.NET die de ontwikkelaar helpt met het maken van zo’n taak. Een trigger wordt gebruikt om duidelijk te maken wanneer de taak moet worden uitgevoerd. Ten slotte is er de planner, die ervoor zorgt dat de taak uitgevoerd wordt aan de hand van het vooraf gedefinieerde schema. [22] Anjan Kant, een manager van een multidisciplinair team bij Infotec Global Services, geeft hier nog wat extra informatie over. Een taak kan een naam krijgen van de ontwikkelaar en kan vervolgens aan een relevante groep toegevoegd worden. Hetzelfde geldt voor triggers. Dit komt omdat iedere taak maar één keer in de planner geregistreerd kan worden, maar een taak kan wel geregistreerd worden met verschillende triggers. [25]

### Prijs

Er wordt niet veel vermeld over de prijs van Quartz.NET. Het is een opensourceframework. De *source*-code is volledig te downloaden op GitHub. Dit betekent dat Quartz.NET gratis is om te gebruiken, zegt Ilya Chumakov. [20]

### Complexiteit

Het grote probleem met Quartz.NET is dat het gewoon te complex is, zegt Thomas Hansen. De syntax is gebaseerd op die van configuratiebestanden. Dit zorgt ervoor dat de code soms zelfs te complex is om door de Integrated Development Environment (IDE) automatisch aangevuld te worden. Bovendien zijn er te veel regels code nodig om een eenvoudige taak te maken. [21] Volgens Yogesh R, een Microsoft Dynamic CustomerRelationshipManagement (CRM)-consultant met meer dan tien jaar ervaring, is Quartz.NET juist wel eenvoudig opgebouwd en wordt het gebruikt in complexe applicaties. [26]

### Opslagmogelijkheden

Quartz.NET biedt een *persistence*-feature aan. Hiermee kunnen de taken opgeslagen worden in een database. Dit betekent dat na het afsluiten van de applicatie alle taken nog altijd behouden blijven. Quartz.NET biedt hier een aantal providers aan, out of de box, zoals SQL Server, PostgreSQL of SQLite. [22] Ilya Chumakov vermeldt ook nog de mogelijkheid om gebruik te maken van MySQL, Oracle en zelfs In-memory. Bij In-memory wordt de data niet bewaard als de applicatie uitvalt. [20]

## Azure Logic Apps

### Triggers, acties en connectors

In Azure Logic apps zijn er drie componenten waarmee de gebruiker workflows kan creëren. Deze componenten zijn triggers, acties en connectors. Kristof D’Hoossche licht deze kort toe. De trigger is het startpunt van de workflow. Deze geeft aan de applicatie of service door wanneer een specifieke gebeurtenis plaatsvindt. Het gevolg hiervan is dat de workflow gestart wordt. Nadat de trigger geactiveerd is, wordt een bepaalde actie uitgevoerd. Een actie verwijst naar een bewerking die wordt gedefinieerd door een connector. Connectors leggen een verbinding tussen de applicatie en software van derden. Azure Logic apps biedt veel connectors aan, out-of-the-box, een aantal voorbeelden hiervan zijn Dynamics 365, Office 365, Dropbox, Twitter en nog veel meer. [16]

Anji Velagana, een Content Lead bij MindMajix, geeft extra uitleg over triggers en connectors: [27]

#### Triggers

Er zijn drie soorten triggers in Azure Logic Apps:

* Poll triggers: deze worden geactiveerd wanneer er nieuwe gegevens beschikbaar zijn.
* Push triggers: deze triggers luisteren naar gegevens op een endpoint, dus de trigger wordt aangesproken wanneer een gebeurtenis plaatsvindt. Hierdoor wordt de workflow opnieuw uitgevoerd.
* *Recurrence* trigger: dit zijn triggers die geactiveerd worden op een bepaald schema.

#### Connectors

Er zijn een aantal soorten connectors in Azure logic Apps:

* Standard connectors: deze zitten standaard in Azure Logic Apps. Service Bus, Power BI, Oracle-Database, OneDrive enzovoort zijn een aantal voorbeelden hiervan.
* *Integration* account connectors: deze zijn enkel beschikbaar met een *Integration* account. Met behulp van deze connectors kan Extensible Markup Language (XML) getransformeerd en gevalideerd worden. Ook kunnen business-to-business berichten verwerkt worden met AS2/X12/EDIFACT. Dit zijn *flat*-files die gebruikt kunnen worden door twee bedrijven om op een veilige manier zakelijke transacties en bestanden te verzenden en te ontvangen.
* *Enterprise* connectors: deze geven support voor International Business Machines (IBM), Message Queue (MQ) en Systemen, Applicaties en Producten (SAP). Hiervoor moet wel extra betaald worden.

### Design tool

De designtool van Azure Logic Apps is eenvoudig om te gebruiken en bespaart veel tijd van de ontwikkelaar. Het visuele aspect van de designer van Azur Logic Apps is een gestructureerde en duidelijke oplossing om grote, meer complexe workflows te creëren maar het kan ook gebruikt worden om snel een eenvoudige workflow te maken. De designer biedt veel mogelijkheden om workflows te maken. Als er een actie nodig is die niet bestaat in Azure Logic Apps, kan altijd nog een REST API gebruikt worden die aangesproken wordt met behulp van Azure Logic Apps. Ten slotte is de designer ook in de browser beschikbaar, dus de ontwikkelaar kan van overal nieuwe workflows maken of oude workflows aanpassen. Dit wordt bevestigd door zowel Anji Velagana als CMarix, een groot outsourcing bedrijf gespecialiseerd in het maken van *enterprise* software, mobile applicaties, websites en e-commerce. [27] [28]

### Prijs

De prijs van Azure Logic Apps is ook een positief gegeven, zeggen Anji Velagana en CMarix. Er is namelijk geen vast abonnement dat betaald moet worden. De gebruiker betaalt voor wat hij verbruikt. Hiermee wordt bedoeld dat hoe vaker de Logic App gebruikt wordt, hoe meer de gebruiker betaalt. Dit is goed voor kleine applicaties want dan is het ook veel goedkoper, bijvoorbeeld als de gebruiker een DropBox-connector heeft met een trigger voor gewijzigde bestanden, kost dit de gebruiker niets tenzij er effectief bestanden gewijzigd worden. [27] [28]

### Uitbreidbaarheid

Azure Logic Apps is heel uitbreidbaar, zegt Anji Velagana, omdat iedere trigger een nieuwe LogicApp zal instantiëren. Dit zorgt ervoor dat triggers die op hetzelfde moment geactiveerd worden niet dezelfde LogicApp aanspreken. Het gevolg hiervan is dat de taak sneller klaar is. [27] Ook is de LogicApp herbruikbaar. Dit is handig omdat de ontwikkelaar niet twee keer dezelfde LogicApp moet aanmaken. Hij kan gewoon de eerder gemaakte LogicApp hergebruiken. [28]

# Proof of Concept

In dit deel van het onderzoek wordt er een PoC gemaakt met iedere tool. In deze PoC wil de gebruiker een taak plannen op een bepaald schema. De gebruiker kan dit schema kiezen aan de hand van een dropdownbox. Door op een knop te drukken wordt de taak opgeslagen in een database zodat de taak bijgehouden wordt, zelfs als de applicatie uit valt. Aangezien ervan uitgegaan wordt dat de gebruiker niet zomaar een databaseserver heeft moet daar een andere oplossing voor gezocht worden. In het geval van deze PoC is er gekozen om gebruik te maken van SQLite-databases. Aan de hand van deze PoC wordt er nagegaan of alles wat in de literatuurstudie vermeld is, wel accuraat is.

## Hangfire

### Installatie

Voor het installeren van Hangfire zijn er minstens drie projecten nodig, namelijk een project voor de frontend, het Hangfire-serverproject en een project voor de taak die moet worden uitgevoerd. Voor zowel de frontend als de Hangfire-server zijn er een aantal NuGet-packages nodig. De Hangfire NuGet-package is daar één van. Daarbovenop is er ook een extensie geïnstalleerd, namelijk Hangfire.Storage.SQLite. Deze package is nodig om de aangemaakte taken op te slaan in een SQLite- database. Hangfire.Storage.SQLite is afhankelijk van drie andere packages, namelijk Microsoft.Data.Sqlite, System.Data.SQLite en System.Data.SQLite.EF6. Zonder deze drie packages kan data niet opgeslagen worden in de database.

Het uitzoeken van dit systeem is niet zo gemakkelijk. Maar er is wel een Pluralsight-cursus die de ontwikkelaar uitlegt hoe dit gedaan moet worden. De Pluralsight-cursus noemt “Simplified .NET Background Tasks with Hangfire”. In deze cursus wordt uitgelegd hoe Hangfire gebruikt kan worden met een ASP.NET-applicatie maar alles blijft bijna hetzelfde in WPF-applicaties. Er wordt niet ingegaan over hoe er gewerkt moet worden met een SQLite database.

### Configuratie

Het installeren van Hangfire is het moeilijkste gedeelte. Het configureren van de Hangfire-server daarentegen is heel eenvoudig. Om de Hangfire-server te configureren moet er eerst een databaseprovider gekozen worden, in dit geval is dat SQLite. Hiervoor moet enkel de GlobalConfiguration van Hangfire aangepast worden. Vervolgens moet de BackgroundJobServer van Hangfire geïnstantieerd worden. Deze zal regelmatig de database controleren op taken die moeten uitgevoerd worden.

### Taak toevoegen

Een taak toevoegen in Hangfire is eenvoudig. Elke soort taak kan toegevoegd worden met maar één regel code. Aangezien voor deze PoC het de bedoeling was dat de taak op een bepaald schema wordt uitgevoerd, kan hiervoor gebruikgemaakt worden van de RecurringJob in Hangfire. Deze heeft een AddOrUpdate-methode die een RecurringJob zal toevoegen of aanpassen in de database. Deze methode accepteert twee parameters, namelijk de taak die moet worden uitgevoerd aan de hand van een functie en een cron-expressie. In het geval van deze PoC wordt er hier dus een pingcommando uitgevoerd. Dit wordt gedaan op de volgende manier: RecurringJob.AddOrUpdate(() = > PingJob.Ping(), Cron.Daily);. In het voorgaande voorbeeld wordt een Cron.Daily meegegeven. Dit is een Enum die aangeboden wordt door Hangfire. Deze kan afhankelijk van de input veranderd worden. Zo is er in deze PoC een switch gemaakt die afhankelijk van de input een Cron-expressie kiest.

## Quartz.NET

### Installatie

In tegenstelling tot Hangfire is de installatie van Quartz.NET zeer eenvoudig. De enige NuGet-package die nodig is, is die van Quartz.NET omdat Quartz.NET geen extensies nodig heeft. Alle functionaliteit van Quartz.NET zit in de originele package. Er is wel nog één package nodig om data weg te kunnen schrijven naar de SQLite-database, namelijk System.Data.SQLite.Core.

### Configuratie

Quartz.NET werkend krijgen met een *InMemoryDatabase* is heel eenvoudig. Nadat de NuGet-package geïnstalleerd is, moet er enkel nog een *Scheduler* aangemaakt worden en om dit te doen zijn er maar twee regels code nodig. Maar vanaf het moment dat er gebruikgemaakt moet worden van een database begint het al moeilijker te worden. Er moeten namelijk in de App.config-file een aantal *Key-Value pairs* toegevoegd worden. Dit is wel vermeld in de documentatie maar er staat niet uitgelegd hoe dit moet. Een snelle Google-search legt dit wel uit. Dit geldt ook voor een aantal andere onderwerpen. De documentatie is onvoldoende gedetailleerd. Een ander gedeelte dat niet zo duidelijk wordt uitgelegd in de documentatie is dat de tabellen van de database niet aangemaakt worden door Quartz.NET. De ontwikkelaar moet deze zelf aanmaken. In Hangfire gebeurt dit automatisch.

### Taak toevoegen

Een taak toevoegen in Quartz.NET is redelijk eenvoudig. Eerste moet er een taakklasse gemaakt worden die overerft van de IJob interface van Quartz.NET. In deze klasse zit een Execute-methode. In deze methode wordt de taak uitgevoerd. In het geval van deze PoC wordt hier dus een pingcommando uitgevoerd. Vervolgens moet er een Job aangemaakt worden. Dit kan aan de hand van de JobBuilder-klasse van Quartz.NET, dit ziet er zo uit JobBuilder.Create<PingJob>().WithIdentity("myJob", "group1").Build();. Het grote verschil met Hangfire is dat er een identiteit moet worden meegegeven omdat een job maar één keer aangemaakt mag worden. Nadat de job aangemaakt is, moet er nog een trigger aangemaakt worden. Dit gebeurt op ongeveer dezelfde manier. Nu wordt er gebruikgemaakt van de TriggerBuilder, bijvoorbeeld TriggerBuilder.Create().WithIdentity("myTrigger", "group1").WithSchedule(CronScheduleBuilder.DailyAtHourAndMinute(0, 0)).Build();. Het enige verschil tussen dit en de JobBuilder is dat er ook een voorwaarde moet voorzien worden die de trigger zal activeren. In deze PoC is gekozen om gebruik te maken van een schema aan de hand van een cron-expressie. Ten slotte moeten de job en de trigger toegevoegd worden aan de *Scheduler*. Zoals in Hangfire heeft Quartz.NET ook een object dat gebruikt kan worden om een Cron-expressie op een eenvoudige manier te genereren, namelijk de CronScheduleBuilder. Deze bevat een aantal methodes om Cron-expressies te berekenen.

## Azure Logic Apps

### Registratie

Azure Logic Apps is anders dan Hangfire en Quartz.NET, het is namelijk geen framework maar een onlinetool. Dit heeft als gevolg dat de applicatie niet geïnstalleerd moet worden en dat de tool overal beschikbaar is. Om de tool te kunnen gebruiken is er wel internetverbinding nodig. Om Azure Logic Apps te gebruiken moet er een Azure-account aangemaakt worden. In het geval van deze PoC is er gebruikgemaakt van een studentenaccount. Een student ontvangt een tegoed van honderd dollar en gratis ontwikkelprogramma’s. Maar zoals eerder in dit eindwerk vermeldt, kost Azure Logic Apps pas als de LogicApp gebruikt wordt.

### Voorbereiding

Om Azure Logic Apps te gebruiken moet er niet veel geconfigureerd worden. Het enige dat moet gebeuren is een nieuwe LogicApp aanmaken. Dit is eenvoudig via het Azure-Portaal. Er is wel een probleem: Azure Logic Apps kan niet zomaar een opdrachtprompt-commando uitvoeren en dit is wel één van de belangrijkste criteria. Hier zijn twee oplossingen voor.

Er kan gebruikgemaakt worden van een REST API die *requests* kan ontvangen om een opdrachtprompt-commando uit te voeren. De LogicApp kan dan met een bepaalde trigger een *request* sturen naar de REST API. Dit is echter geen goede oplossing aangezien ervan uitgegaan wordt dat de gebruiker geen REST API kan hosten.

De tweede oplossing is dat de ontwikkelaar TRIGGERcmd installeert. Dit is een programma dat opdrachtprompt-commando’s kan uitvoeren vanop afstand. Er zit standaard een actie van TRIGGERcmd in Azure Logic Apps. De actie moet gekoppeld worden aan een TRIGGERcmd-account. Ten slotte moet de TRIGGERcmd-Agent geïnstalleerd worden op de computer waarop het opdrachtprompt-commando moet worden uitgevoerd. Er is gekozen om de PoC te maken op deze manier omdat het voldoet aan alle voorwaarden.

### Taak aanmaken

Een taak aanmaken met Azure Logic Apps is redelijk eenvoudig met de visuele designtool. Het eerste dat nodig is om een taak te maken, is een trigger die de taak zal starten volgens een gegeven schema. Dit is gemakkelijk. Azure Logic Apps biedt namelijk een Schedule-trigger aan met een Recurrence-optie waar een interval- en frequentie moet worden ingevuld. Vervolgens moet er in de TRIGGERcmd-Agent een nieuwe taak aangemaakt worden die een ping-commando uitvoert. Dit commando moet dan toegevoegd worden als trigger in de Logic App. Dit is ook weer redelijk eenvoudig om te doen aangezien TRIGGERcmd-acties ondersteund worden door Azure Logic Apps. Hiervoor zijn twee invulvelden, namelijk computer en *command*. Om dit te kunnen doen moet de Logic App gekoppeld worden met een TRIGGERcmd-account. Dit vraagt Azure Logic Apps wanneer de actie wordt toegevoegd. Het is wel zo dat dit systeem totaal niet gebruiksvriendelijk is voor gebruikers met weinig kennis over computers.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Figuur 31 Logic App met een schema als trigger en ping-commando als actie

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Figuur 32 TRIGGERcmd-Agent

# Conclusie

## Voor- en nadelen

Om erachter te komen welke optie het beste is voor deze stageopdracht wordt er een lijst van voor- en nadelen gemaakt. De informatie in deze lijst komt zowel uit de literatuurstudie als de PoC’s.

### Hangfire

|  |  |
| --- | --- |
| **Voordelen** | Nadelen |
| * Gratis * Uitgebreide documentatie * Eenvoudig te gebruiken * Pluralsight-cursus * Veel uitbreiding via Nuget-packages | * Iets ingewikkeldere installatie door NuGet |

Tabel 1 Voor- en nadelen Hangfire

### Quartz.NET

|  |  |
| --- | --- |
| **Voordelen** | Nadelen |
| * Gratis * Pluralsight-cursus * Alle functionaliteiten zitten in één NuGet-package * Syntax gebaseerd op configuratiebestanden | * Syntax gebaseerd op configuratiebestanden * Niet zo gedetailleerde documentatie |

Tabel 2 Voor- en nadelen Quartz.NET

### Azure Logic Apps

|  |  |
| --- | --- |
| **Voordelen** | Nadelen |
| * Geen vast abonnement * Online * Veel ondersteuning voor software van derden * Geen installatie nodig * Geen programmatie nodig | * Internetverbinding is vereist * Niet gebruiksvriendelijk voor gebruikers met weinig computerkennis * Niet gratis |

Tabel 3 Voor- en nadelen Azure Logic Apps

## Welke tool/framework is de beste oplossing voor deze opdracht?

Het grote probleem met Azure Logic Apps is dat het niet echt gebruiksvriendelijk is voor gebruikers met weinig computerkennis. Gebruiksvriendelijkheid is één van de belangrijkste criteria van deze stageopdracht. Quartz.NET en Hangfire daarentegen zijn beiden wel goede oplossingen om een gebruiksvriendelijke takenplanner te maken omdat de ontwikkelaar de applicatie volledig zelf kan maken. Ze zijn gratis, hebben ondersteuning voor SQLite en ze hebben beiden geen webserver nodig om te werken. Door de uitgebreide documentatie van Hangfire is deze iets gemakkelijker voor de ontwikkelaar om te leren en kan er geconcludeerd worden dat Hangfire de betere oplossing is. Quartz.NET is zeker geen slechte oplossing.

1. Reflectie

De stageopdracht zelf, een tool maken om taken te automatiseren, was heel interessant. Het is een eenvoudig concept waar toch meer komt bij kijken dan je eerst verwacht. Ik heb gebruikgemaakt van allerlei technologieën waar ik nog nooit van gehoord had, zoals bijvoorbeeld Hangfire, Prism, enzovoort. Maar datgene dat ik vooral heel interessant vond aan deze stageopdracht is de analysefase. Op school was het altijd zo dat je een opdracht kreeg en die moest opgelost worden met bepaalde technologieën die door de lectoren gekozen werden. Dit was tijdens dit project natuurlijk niet het geval. Ik moest zelf uitzoeken welke frameworks ik ging gebruiken. Dit deed ik samen met de hulp van mijn bedrijfspromotor, Kris Vanreyten, en de rest van het team. Het ontwikkelproces kwam natuurlijk ook met zijn uitdagingen. Maar dat vind ik juist fijn aan programmeren.

Tijdens deze stage zijn er een aantal zaken waarvan ikzelf, mijn bedrijfspromotor, de klant en de rest van het team vinden dat ik het goed heb gedaan. Eén van deze zaken is het contact met de klant. Tijdens mijn stage had ik heel veel contact met de klant. We hadden twee keer per week een scrummeeting. Op deze manier kon ik vragen stellen, kon ik tonen wat ik de laatste paar dagen gedaan had en zij konden hierop feedback geven. Soms waren er ook meetings waarbij ik samen moest zitten met een ontwikkelaar van de klant. Mijn bedrijfspromotor zat vaak ook in deze meetings, voor ondersteuning, maar soms moest ik ze alleen doen. Ook heb ik een presentatie/demo moeten geven aan een aantal stakeholders van de klant. Aangezien een aantal van de stakeholders geen Nederlands sprak en een ander gedeelte geen Frans, heb ik deze in het Engels gedaan. Deze verliep ook heel goed en ik heb na de presentatie een aantal berichtjes gekregen van de stakeholders dat ik het goed had gedaan. Dit is altijd fijn om te horen. Ook het proces van analyse tot voltooiing vind ik dat goed verlopen is. Zoals ik eerder vermeld heb, is het hele analyseproces iets dat ik nog nooit heb moeten doen maar ik vind wel van mezelf dat ik dit op een goede manier gedaan heb. Ten slotte is er gezegd tijdens de overdrachtsmeeting, een meeting met een aantal ontwikkelaars van de klant, dat mijn code er heel gestructureerd uitziet.

Natuurlijk zijn er tijdens de stage ook een aantal zaken die wat minder goed gegaan zijn en die ik meeneem naar de toekomst. Eén van deze zaken is mijn planning. Ik vond het heel moeilijk om tijdens de eerste week al een planning te maken voor de volgende 12 weken. Ik heb dit nog maar een paar keer moeten doen tijdens de les van Business Advanced en hierbij kreeg je de taken die allemaal moesten gebeuren. Dat was niet het geval bij deze opdracht. De planning die ik uiteindelijk gemaakt heb, was een redelijk goede inschatting van wat ik ging doen maar de duur van deze taken klopte niet. Sommige taken duurden langer dan ik had ingeschat en sommige korter. Dit is natuurlijk iets dat beter zal gaan door ervaring en hier ga ik zeker op oefenen. Ook het schrijven van testen kan beter. Tijdens mijn IT-project op de PXL heb ik al mijn testen tot het laatste moment bewaard. Dit was heel slecht want er moest veel getest worden en uitgezocht worden hoe je die dingen test. Toen had ik gezegd dat ik nooit meer al mijn testen tot het laatste ging bewaren. Tijdens dit project is dit al veel beter gegaan, ik testte meestal iedere keer als ik een aantal features af had. Maar ik vind zelf dat dit nog beter kan. Het hoeft niet *test driven development* te zijn maar toch kan ik best na iedere feature testen schrijven.

Ik heb uiteraard ook veel bijgeleerd tijdens deze stageopdracht. Op programmeervlak heb ik vooral gewerkt met nieuwe frameworks waarvan ik nu kan zeggen dat ik ze redelijk goed ken. Onze weg vinden met nieuwe technologieën is iets waar de lectoren heel veel moeite in steken. Het maken van een gebruiksvriendelijke applicatie kwam tijdens mijn studies minder aan bod. Dit is wel iets waar ik veel over heb bijgeleerd. Het uitzoeken van hoe je je applicatie gebruiksvriendelijk maakt is niet gemakkelijk, maar dit is nog maar de helft van het werk. Je moet het namelijk ook nog overzetten naar XAML-code. Dit is ook iets dat ik eerder zelf heb moeten uitzoeken. Maar ik vind wel dat dit redelijk goed gelukt is.

Er zijn ook een aantal zaken waar ik trots op ben. Eerst en vooral ben ik trots op mijn eindwerk. Ik heb dyslexie en ben helemaal niet goed in teksten schrijven. Maar ik heb mijn best gedaan en ik vind dat het resultaat toch wel een mooi verhaal is geworden. Dit had ik niet kunnen doen zonder mijn hogeschoolpromotor, Carine Derkoningen en mijn bedrijfspromotor. Ook ben ik trots op mijn contract bij Tobania. Dit is toch wel een teken dat ze tevreden zijn over mijn werk en dat ze potentieel in mij zien.

Tijdens deze periode heb ik ook onderzoek moeten doen naar een aantal frameworks/tools. Ik moest bepalen welk van deze frameworks de beste oplossing was voor deze stageopdracht. Dit heb ik gedaan door eerst een aantal bronnen over de frameworks/tools met elkaar te vergelijken. Vervolgens heb ik zelf een PoC gemaakt met iedere framework/tool om voor mijzelf te ervaren hoe deze frameworks/tools in elkaar zitten. Uit deze literatuurstudie en PoC’s heb ik de voor- en nadelen van iedere framework/tool gehaald om zo te concluderen welke framework/tool het beste was.

Voor de stageopdracht is gebruik gemaakt van Hangfire. Na het onderzoek bleek dat dit zeker de beste oplossing was, vooral omdat de klant al ervaring had met Hangfire. Maar tijdens dit onderzoek ben ik nog een tool tegengekomen die hiervoor gebruikt zou kunnen worden, namelijk Microsoft Power Automate. Was deze tool nog beter geweest? Hier kan misschien nog verder onderzoek naar gedaan worden.

In het begin van dit onderzoek was ik onzeker of dit wel ging lukken. Ik wist niet goed hoe ik eraan moest beginnen. Er waren maar weinig bronnen beschikbaar over de frameworks/tools en al helemaal geen die de drie met elkaar vergeleken. Hier heeft mijn hogeschoolpromotor mij goed op weg geholpen door veel feedback te geven. Het maken van de PoC’s lukte beter. Dit is namelijk programmeren en hier ben ik goed in.

Over het algemeen ben ik tevreden over mijn bachelorproef en kijk ik ernaar uit om bij Tobania te kunnen werken.

Bronnenlijst

|  |  |
| --- | --- |
| [1] | „EverybodyWiki,” [Online]. Available: https://nl.everybodywiki.com/Tobania. [Geopend 09 04 2021]. |
| [2] | „Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/.NET\_Core. [Geopend 15 04 2021]. |
| [3] | „Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/Windows\_Presentation\_Foundation. [Geopend 15 04 2021]. |
| [4] | „What is Entity Framework,” [Online]. Available: https://www.entityframeworktutorial.net/what-is-entityframework.aspx. [Geopend 15 04 2021]. |
| [5] | „Topshelf.ReadTheDocs,” [Online]. Available: https://topshelf.readthedocs.io/en/latest/overview/index.html. [Geopend 15 04 2021]. |
| [6] | „GitHub,” [Online]. Available: https://github.com/serilog/serilog. [Geopend 15 04 2021]. |
| [7] | „PrismLibrary,” [Online]. Available: https://prismlibrary.com/docs/index.html. [Geopend 01 05 2021]. |
| [8] | „GitHub,” [Online]. Available: https://github.com/nunit/nunit. [Geopend 01 05 2021]. |
| [9] | „Wikipedia,” [Online]. Available: https://en.wikipedia.org/wiki/NUnit. [Geopend 01 05 2021]. |
| [10] | „GitHub,” [Online]. Available: https://github.com/moq/moq4. [Geopend 01 05 2021]. |
| [11] | F. Verhelst, „AgileScrumGroup,” [Online]. Available: https://agilescrumgroup.nl/wat-is-een-user-story/. [Geopend 07 04 2021]. |
| [12] | D. d. Vries, „frankwatching,” 28 05 2019. [Online]. Available: https://www.frankwatching.com/archive/2019/05/28/5-basistips-om-de-ux-van-je-website-te-verbeteren/. [Geopend 23 05 2021]. |
| [13] | H. Maltha. [Online]. Available: https://www.frankwatching.com/archive/2009/11/23/wireframes-de-bouwtekening-van-een-website/. [Geopend 14 04 2021]. |
| [14] | „Capterra,” Talkdesk, [Online]. Available: https://www.capterra.be/software/132852/talkdesk#about. [Geopend 22 05 2021]. |
| [15] | „support.talkdesk,” Talkdesk, [Online]. Available: https://support.talkdesk.com/hc/en-us/articles/115001284103-Talkdesk-APIs. [Geopend 22 05 2021]. |
| [16] | K. D'Hoossche, „Net-It,” 02 05 2018. [Online]. Available: https://www.net-it.be/azure-logic-apps/. [Geopend 22 04 2021]. |
| [17] | M. Murugan, „CodeWithMukesh,” 27 05 2020. [Online]. Available: https://codewithmukesh.com/blog/hangfire-in-aspnet-core-3-1/. [Geopend 14 04 2021]. |
| [18] | J. Kanjilal, „InfoWorld,” 01 04 2016. [Online]. Available: https://www.infoworld.com/article/3050769/how-to-work-with-hangfire-in-c.html. [Geopend 14 04 2021]. |
| [19] | „Hangfire,” [Online]. Available: https://docs.hangfire.io/en/latest/. [Geopend 29 04 2021]. |
| [20] | I. Chumakov, „CodingSight,” 01 06 2017. [Online]. Available: https://codingsight.com/hangfire-task-scheduler-for-net/. [Geopend 14 04 2021]. |
| [21] | T. Hansen, „DZone,” 28 11 2019. [Online]. Available: https://dzone.com/articles/a-dynamic-task-scheduler-for-aspnet-core. [Geopend 14 04 2021]. |
| [22] | J. Kanjilal, „InfoWorld,” 03 06 2016. [Online]. Available: https://www.infoworld.com/article/3078781/how-to-work-with-quartz-net-in-c.html. [Geopend 14 04 2021]. |
| [23] | [Online]. Available: https://en.wikipedia.org/wiki/Cron. [Geopend 14 04 2021]. |
| [24] | J. D. Ruysscher, „ForEach,” 28 11 2019. [Online]. Available: https://www.foreach.be/blog/the-why-and-how-of-hangfire?lang=en. [Geopend 16 04 2021]. |
| [25] | A. Kant, „TechnologyCrowd,” [Online]. Available: https://www.technologycrowds.com/2021/03/open-source-quartz-job-scheduler-for-dot-net.html. [Geopend 16 04 2021]. |
| [26] | Y. R., „Cisin,” [Online]. Available: https://www.cisin.com/coffee-break/Enterprise/take-advantage-of-a-quartz-net-hosted-service-to-schedule-background-jobs-in-your-asp-net-core-application.html. [Geopend 16 04 2021]. |
| [27] | A. Velagana, „MindMajix,” 30 03 2021. [Online]. Available: https://mindmajix.com/overview-of-azure-logic-apps. [Geopend 06 05 2021]. |
| [28] | cmatrix, 22 10 2019. [Online]. Available: https://www.cmarix.com/blog/azure-logic-apps-how-it-works-and-benefits-developers/. [Geopend 06 05 2021]. |