Technical design document

In dit document zal ik de technische keuzes en aspecten toelichten. De verschillende foundation en cornerstone modules zullen besproken en uitgewerkt worden in dit document.

# Enterprise Software Platforms

## Criteria

Er worden 3 verschillende criteria gebruikt om het beste framework te vinden voor een project:

- **Range of Applicability** (welk deel van de applicatie wordt ontworpen met een bepaald

framework? Moet je er één of meer gebruiken?

- **Development Speed** (hoe snel wil je de applicatie ontwikkelen? Hoe onderhoud je het?)

- **Manageability and Flexibility** (wat zijn de limieten van een framework?)

Bij het ontwerpen van de applicatie staat de backend centraal. Bij het maken van de backend zullen de grootste uitdaging zitten, het front-end moet alleen functioneel zijn om aan te tonen dat de backend werkt.

Omdat dit project vooral om het leerproces gaat meer dan de applicatie zelf is het verstandig om een framework te kiezen met veel flexibiliteit. De onderwerpen die aan bod komen zijn nog onbekend voor mij en weet niet wat een framework nodig heeft om succesvol de applicatie te kunnen draaien. Doordat er een groot leerproces achter het project zit zal de development speed ook lager liggen dan de norm.

Het leerproces is een belangrijk onderdeel van dit project. Daarom lijkt het mij verstandig om een framework te kiezen die past bij de criteria die ik net heb opgesteld en waarmee ik al eerder heb gewerkt. Om onnodige leerprocessen te vermijden.

Zo blijven er twee frameworks over die ik kan gebruiken voor mijn backend: .NET Core en Java EE. Deze twee frameworks zijn bijna identiek aan elkaar. Meeste functies die .NET biedt zitten ook in Java. Mijn voorkeur gaat uit naar .NET Core aangezien ik daar meer ervaring mee heb dan Java.

Voor het front-end zijn er een aantal opties: Node.js, React, Vue,js, Angular, JavaScript, HTML, PHP en Svelte. Het front-end is niet een belangrijk onderdeel van dit project, dus er is geen specifiek beste optie. Daarom heb ik het framework gekozen dat mij het meeste aanspreekt: Svelte. Svelte is een erg jong nieuw framework dat (van wat ik heb gelezen) erg fijn is om mee te werken. Het is gebaseerd op JavaScript dus ik zal niet compleet blind in het nieuwe framework stappen.

Er is nog één optioneel framework dat ik ga gebruiken. Om communicatie tussen de backend en een database makkelijk te maken ga ik gebruik maken van Entity Framework Core. Een ORM. Dit maakt het mogelijk om met .NET objecten te werken en deze in de database te zetten.

# Architectuur stijl onderzoek

**Welke architectuur soort is het best toe te passen op het project?**

**Methods: Competitive Analysis/Available product analysis – Literature study**

Verschillende beschikbare architecturen bekeken en tegenover elkaar gezet

Uiteindelijk op basis van een matrix gekeken welk architectuur het best bij het project past.

**Een aantal Architectuur soorten en hun pros en cons/functionaliteiten:**

|  |  |  |
| --- | --- | --- |
| Architectuur | Gebruik | Tekortkomingen |
| Layered Architecture | Applicaties die snel gebouwd moeten zijn | Ongeorganiseerde broncode en modules met geen definitieve rollen kunnen een probleem vormen |
|  | Enterprise applicaties die traditionele IT-divisies en processen nodig hebben | Het overslaan van eerdere lagen om een strakke koppeling te creëren, kan leiden tot een logica puinhoop vol complexe onderlinge afhankelijkheden |
| Handig voor teams met weinig ervaring die niet veel weten over architectuur patronen | Basisaanpassingen kunnen een volledige redeployment van de applicatie vereisen |
| Applicaties die een strenge standaard aan onderhoud en testability nodig hebben |  |

|  |  |  |
| --- | --- | --- |
| Architectuur | Gebruik | Tekortkomingen |
| Event-driven Architecture Pattern | Voor applicaties waar afzonderlijke gegevensblokken samenwerken met maar enkele modules. | Het testen van individuele modules kan alleen worden gedaan als ze onafhankelijk zijn, anders moeten ze worden getest in een volledig functioneel systeem |
|  | Helpt met user interfaces | Wanneer meerdere modules dezelfde events verwerken, wordt error handling een uitdaging om te structureren. |
|  | | De ontwikkeling van een systeembrede gegevensstructuur voor gebeurtenissen kan moeilijk worden als de events verschillende behoeften hebben. |
| Aan te raden voor een high performant Agile project |

|  |  |  |
| --- | --- | --- |
| Architectuur | Gebruik | Tekortkomingen |
| Microkernel Architecture Pattern | Toepassingen met een duidelijke segmentatie tussen basisroutines en regels van hogere orde | De plug-ins moeten goede handshaking-code hebben, zodat de microkernel op de hoogte is van de installatie van de plug-in en klaar is om te werken |
|  | Toepassingen met een vaste set kernroutines en een dynamische set regels die regelmatig moeten worden bijgewerkt | Het veranderen van een microkernel is bijna onmogelijk als er meerdere plugins van afhankelijk zijn. |
|  | | Het is moeilijk om vooraf de juiste granulariteit (Hoe deelbaar een systeem is) voor de kernelfunctie te kiezen en later is het complexer. |

|  |  |  |
| --- | --- | --- |
| Architectuur | Gebruik | Tekortkomingen |
| Microservices Architecture Pattern | Bedrijven en webapplicaties die een snelle ontwikkeling vereisen. | Het ontwerpen van het juiste granulariteitsniveau voor een servicecomponent is altijd een uitdaging. |
|  | Websites met kleine componenten, datacenters met goed gedefinieerde grenzen en teams op afstand wereldwijd. | Alle toepassingen bevatten geen taken die kunnen worden opgesplitst in onafhankelijke eenheden. |
|  | | De prestaties kunnen worden beïnvloed doordat taken worden verspreid over verschillende microservices. |

|  |  |  |
| --- | --- | --- |
| Architectuur | Gebruik | Tekortkomingen |
| Space-Based Architecture | Applicaties en softwaresystemen die functioneren met een groot gebruikersbestand en een constante lading aanvragen. | Het is een complexe taak om de gegevens snel in de cache op te slaan zonder meerdere kopieën te storen. |
|  | Toepassingen die verondersteld worden schaalbaarheids- en gelijktijdigheidsproblemen aan te pakken. |  |

|  |  |  |
| --- | --- | --- |
| Architectuur | Gebruik | Tekortkomingen |
| Client-Server Architecture | Applicaties zoals e-mails, online bankdiensten, het World Wide Web, afdrukken via netwerken, toepassingen voor het delen van bestanden, gaming-apps, etc. | Incompatibele servercapaciteit kan vertragen, waardoor een prestatieknelpunt ontstaat. |
|  | Applicaties die zich richten op real-time services zoals telecommunicatie-apps zijn gebouwd met een gedistribueerde applicatiestructuur. | Servers zijn meestal gevoelig voor een single point of failure. |
|  | Applicaties die gecontroleerde toegang vereisen en meerdere services bieden voor een groot aantal gedistribueerde clients. | Het veranderen van het patroon is een complex en duur proces. |
|  | Een applicatie met gecentraliseerde bronnen en services die over meerdere servers moet worden gedistribueerd. | Serveronderhoud kan een veeleisende en dure taak zijn. |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| Architectuur | Gebruik | Tekortkomingen |
| Master-Slave Architecture | Ontwikkeling van besturingssystemen die mogelijk een architectuur met meerdere processors vereisen. | Falen van de mastercomponent kan leiden tot een verlies van gegevens zonder back-up over de slave-componenten. |
|  | Geavanceerde toepassingen waarbij grotere diensten moeten worden opgesplitst in kleinere componenten. | Afhankelijkheden binnen het systeem kunnen leiden tot een storing van de slave-componenten. |
|  | Toepassingen die onbewerkte gegevens verwerken die zijn opgeslagen op verschillende servers via een gedistribueerd netwerk. | Er kan een toename van overheadkosten zijn als gevolg van het geïsoleerde karakter van de slave-componenten. |
|  | Webbrowsers die multithreading volgen om de responsiviteit te vergroten. |  |

|  |  |  |
| --- | --- | --- |
| Architectuur | Gebruik | Tekortkomingen |
| Pipe-Filter Architecture | Het kan worden gebruikt voor toepassingen die een eenvoudige, eenrichtingsverkeersverwerking en -transformatie mogelijk maken. | Er kan een verlies van gegevens tussen filters zijn als het infrastructuurontwerp niet betrouwbaar is. |
|  | Toepassingen met behulp van tools zoals Electronic Data Interchange en External Dynamic List. | Het langzaamste filter beperkt de prestaties en efficiëntie van de gehele architectuur. |
|  | Ontwikkeling van data compilers die worden gebruikt voor foutcontrole en syntax analyse. | Tijdens de overdracht tussen filters kunnen de overheadkosten voor gegevenstransformatie toenemen. |
|  | Om geavanceerde bewerkingen uit te voeren in besturingssystemen zoals UNIX, waarbij de uitvoer en invoer van programma's in een reeks zijn verbonden. | Het continue transformationele karakter van de architectuur maakt het minder gebruiksvriendelijk voor interactionele systemen. |
|  |  | |

|  |  |  |
| --- | --- | --- |
| Architectuur | Gebruik | Tekortkomingen |
| Broker Architecture Pattern | Gebruikt in message broker software zoals Apache ActiveMQ, Apache Kafka, RabbitMQ en JBoss Messaging. | Lage fouttolerantiecapaciteit |
|  | Voor het structureren van gedistribueerde systemen met ontkoppelde componenten. | Vereist standaardisatie van de servicebeschrijving. |
|  | | De verborgen laag kan de softwareprestaties verminderen. |
| Hogere latentie en vereist meer inspanning bij de implementatie. |

|  |  |  |
| --- | --- | --- |
| Architectuur | Gebruik | Tekortkomingen |
| Peer-to-Peer Architecture | Netwerken voor het delen van bestanden zoals Gnutella en G2 | Geen garantie voor een hoogwaardige service. |
|  | Op cryptocurrency gebaseerde producten zoals Bitcoin en Blockchain. | Het bereiken van robuuste beveiliging is een uitdaging. |
|  | Multimediaproducten zoals P2PTV en PDTP. | De prestaties zijn afhankelijk van het aantal knooppunten dat op het netwerk is aangesloten. |
|  | | Geen manier om een back-up te maken van bestanden of mappen. |
| Mogelijk heeft u een specifieke interface nodig om het bestand te lezen. |

### Architecture Matrix

**De matrix geeft een score aan elke architectuur om zo een goede vergelijking te kunnen maken:**

High Medium Doubt Low

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Agile | Development | Testing | Performance | Scaling |
| Layered |  |  |  |  |  |
| Event-Driven |  |  |  |  |  |
| Microkernel |  |  |  |  |  |
| Microservices |  |  |  |  |  |
| Spaced |  |  |  |  |  |
| Client-Server |  |  |  |  |  |
| Master-Slave |  |  |  |  |  |
| Pipe-Filter |  |  |  |  |  |
| Peer-to-Peer |  |  |  |  |  |

Aangezien schaalbaarheid een grote factor (zelfs een leeruitkomst) is bij het kiezen van een architectuur stijl, is Microkernel als enige echt geen optie. Voor dit project kies in een Microservice architectuur. Ik heb de meeste en makkelijkste toegang tot de microservice documentatie omdat er al veel over te vinden is in canvas.

# C4 modellen

De C4 modellen zijn bedoeld om de developers een beter beeld te geven van de architectuur van de applicatie die je wilt gaan maken. Zo voorkom je dat een developer tijdens het programmeren tegen een architectuur gebaseerd probleem aan loopt.

De C4 modellen bestaan uit vier verschillende diagrammen die elke een beeld geven van de verschillende dieptelagen van de applicatie.

- Systeem context diagram

Dit diagram is de eerste en het simpelste diagram die je maakt. Het context diagram gaat ook niet op de diepte in van de applicatie maar creëert alleen een beeld van de elementen die voorkomen binnen de softwareapplicatie. Dit diagram moet zodanig simpel zijn dat een product owner deze ook kan begrijpen

- Container diagram

Het container diagram laat zien welke architectuur je gaat toepassen op je applicatie. Het diagram laat zien welke functionaliteiten binnen de applicatie aanwezig moeten zijn en waar deze binnen de applicatie staan.

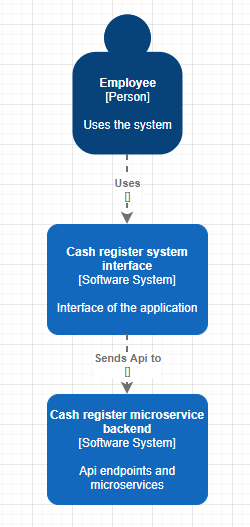
- Component diagram

Een component diagram zoomt in op één container en laat daarvan de onderdelen zien. Het laat de architectuur zien van een container.

- Code diagram

Met een code diagram breng je de code van een individueel component in beeld. Deze diagram is niet altijd nodig als jouw components al genoeg context geven over de code die je wilt gaan schrijven

## System context diagram



Figuur 1 Context diagram

## Container diagram

Figuur 2 Container diagram

## Component diagram

Figuur 3 Component diagram