Ruben Kruiter & Donny Verschuren

  Windesheim Almere AD Software Development

DevOPS Portfolio

Inhoudsopgave

Table of Contents

[Wat is DevOps? 3](#_Toc150156982)

[Waar komt DevOps vandaan? 3](#_Toc150156983)

[Wat was het probleem met deze zogenoemde “silo’s”? 3](#_Toc150156984)

[Wat zijn de voordelen? 3](#_Toc150156985)

[Het belang van documentatie binnen DevOps. 4](#_Toc150156986)

[Kennisoverdracht 4](#_Toc150156987)

[Consistentie en Standaardisatie 4](#_Toc150156988)

[Foutopsporing en Probleemoplossing 4](#_Toc150156989)

[Audit en Compliance 4](#_Toc150156990)

[Verbeterde Samenwerking 4](#_Toc150156991)

[Training en Onboarding 4](#_Toc150156992)

[Beheervormen binnen DevOps 5](#_Toc150156993)

[Beheertaken binnen DevOps 6](#_Toc150156994)

[Het Beheerproces binnen DevOps 6](#_Toc150156995)

[Het belang van documentatie binnen DevOps 7](#_Toc150156996)

[Wat is samenwerken binnen DevOps 7](#_Toc150156997)

[Wat is Hardware? 8](#_Toc150156998)

[De Algemene werking van Hardware binnen een computer? 8](#_Toc150156999)

[Welke hardware zit minimaal in een computer op programma’s te kunnen runnen? 14](#_Toc150157000)

[Hoe zit een netwerk in elkaar? 14](#_Toc150157001)

[De belangrijkste protocollen voor netwerken staan beschreven (minimaal 4) 16](#_Toc150157002)

[De netwerkactiviteiten van een verbinding tussen twee computers staat beschreven 16](#_Toc150157003)

[Een netwerkdiagram tussen eigen computer en server is gegeven 17](#_Toc150157004)

[Er staat beschreven wat Docker is 18](#_Toc150157005)

[Docker VS Virtual Machine 18](#_Toc150157006)

[De procedure om een Dockeromgeving in eigen context te maken staat beschreven 18](#_Toc150157007)

[Een eigen Dockeromgeving staat beschreven 19](#_Toc150157008)

[Foto's van eigen Dockeromgeving worden in draaiende staat getoond 19](#_Toc150157009)

[Er staat beschreven wat acceptatiecriteria zijn 20](#_Toc150157010)

[Van een eigen project staan minimaal vijf acceptatiecriteria beschreven 20](#_Toc150157011)

[Er staan meerdere manieren van testen beschreven 20](#_Toc150157012)

[Op basis van een eigen project is per acceptatiecriterium een test uitgewerkt (inclusief code) 21](#_Toc150157013)

[Er staat beschreven waarom testen belangrijk zijn 24](#_Toc150157014)

[Er zijn minimaal 5 unit tests uitgevoerd op units met een mogelijke return waarde. Deze unittests moeten de goede werking van de units aantonen. 24](#_Toc150157015)

[Er zijn end-to-end test beschreven en uitgevoerd 28](#_Toc150157016)

[In het portfolio wordt in eigen woorden beschreven wat de voor- en nadelen van een Devopspipeline zijn 29](#_Toc150157017)

[Wat is een DevOps – pipeline: 29](#_Toc150157018)

[Voordelen van de DevOps- Pipeline: 29](#_Toc150157019)

[Nadelen van de DevOps-Pipeline: 29](#_Toc150157020)

[In het portfolio wordt in eigen woorden beschreven wat CI-CD inhoudt, onderbouwd met minimaal 3 eigen praktijkvoorbeelden 30](#_Toc150157021)

[In het portfolio is een zelfgeschreven CI-bestand toegevoegd 30](#_Toc150157022)

[Er is een uitgewerkte GIt omgeving opgezet conform het geleerde format 31](#_Toc150157023)

[Het archief met changes en versies toegevoegd van de Git omgeving 31](#_Toc150157024)

# Wat is DevOps?

**DevOps, bestaat uit twee verschillende woorden "Development" en "Operations", en dit werd al snel de afkorting “DevOps”. Maar wat betekent het echt en waarom is het zo revolutionair?**

## Waar komt DevOps vandaan?

DevOps is niet zomaar een afkorting of een set tools; het is een cultuur en een beweging die is ontstaan uit de behoefte om de traditionele barrières tussen softwareontwikkeling (Dev) en IT-operaties (Ops) te doorbreken. In traditionele softwareontwikkelingstrajecten werkten deze twee teams vaak in zogenoemde “silo's”. Dit betekent dat programmeurs hun code schreven en het vervolgens overdroegen aan het operations team voor implementatie. Deze gescheiden aanpak leidde vaak tot vertragingen, miscommunicatie, conflicten en was daarom dus inefficiënt.

## Wat was het probleem met deze zogenoemde “silo’s”?

Deze silo's waren problematisch om verschillende redenen. Ten eerste, zonder constante communicatie en samenwerking, werden fouten vaak pas laat in het ontwikkelingsproces ontdekt, wat leidde tot dure en tijdrovende herzieningen. Ten tweede, omdat de twee teams verschillende doelen en prioriteiten hadden, konden ze soms in conflict raken. Ontwikkelaars waren gefocust op het snel uitbrengen van nieuwe features, terwijl het operations team zich richtte op stabiliteit en uptime. Deze tegenstrijdige doelen leidden vaak tot spanningen.De switch

DevOps is ontstaan als een oplossing voor deze problemen. Het bevordert de samenwerking, communicatie en integratie tussen softwareontwikkeling en IT-operaties. In plaats van in afzonderlijke silo's te werken, werken DevOps-teams samen gedurende de gehele levenscyclus van een applicatie, van ontwerp en ontwikkeling tot implementatie en onderhoud.

## Wat zijn de voordelen?

Het gaat niet alleen om het versneld bouwen van software, hoewel dat een belangrijk voordeel is. DevOps draait ook om het verbeteren van de kwaliteit, betrouwbaarheid en stabiliteit van de applicatie. Door continue integratie, continue levering en geautomatiseerde tests kunnen DevOps-teams sneller en met meer vertrouwen hun werk releasen.

Bovendien, door een DevOps te gebruiken, kunnen organisaties sneller innoveren, sneller reageren op marktveranderingen en een betere klantervaring bieden. Ze kunnen ook profiteren van verhoogde efficiëntie, verminderde kosten en verbeterde teammoraal.

# Het belang van documentatie binnen DevOps.

**Documentatie speelt een belangrijke rol in elk technisch proces, en binnen de context van DevOps is het belang ervan nog meer beduidend. Hier zijn de redenen waarom documentatie essentieel is binnen DevOps:**

## Kennisoverdracht

In DevOps-teams wisselen leden vaak van rol of project. Gedetailleerde documentatie zorgt ervoor dat kennis niet verloren gaat en dat nieuwe teamleden snel inzicht in projecten kunnen krijgen.

## Consistentie en Standaardisatie

Documentatie biedt een standaard aanpak voor processen en procedures. Dit zorgt voor consistentie, vooral wanneer meerdere teams aan verschillende aspecten van een project werken is het handig om deze methode aan te houden.

## Foutopsporing en Probleemoplossing

Wanneer er zich een probleem voordoet, is de eerste waar een ontwikkelaar naar kijkt vaak de documentatie. Goed gedocumenteerde processen en systemen maken het gemakkelijker om fouten te ontdekken en op te lossen.

## Audit en Compliance

Voor organisaties die moeten voldoen aan bepaalde industriestandaarden, helpt documentatie bij het aantonen van compliance tijdens audits.

## Verbeterde Samenwerking

DevOps draait, zoals eerder vermeld, om samenwerking tussen ontwikkeling en operaties. Documentatie biedt een duidelijk referentiepunt voor beide teams, wat miscommunicatie vermindert en samenwerking bevordert.

## Training en Onboarding

Nieuwe teamleden kunnen snel worden getraind en geïntegreerd in het team met behulp van uitgebreide documentatie.

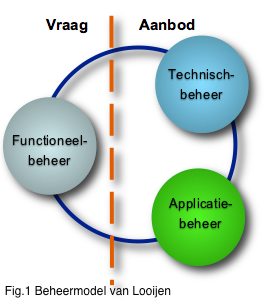
Ik denk dat het nu wel duidelijk is waarom de documentatie binnen DevOps dus zo belangrijk is.

# Beheervormen binnen DevOps

Volgens het beheermodel van Van Looijen zijn er 3 beheervormen

Deze zijn als volgt:

* Functioneel Beheerder
* Technisch Beheerder
* Applicatie Beheerder



Wat deze beheerders doen is als volgt:

De **Functioneel Beheerder** is verantwoordelijk voor het vertalen van de klant naar bedrijfstermen of functies. Daarnaast helpt de Functioneel beheerder de klant of eindgebruiker op weg binnen de gerealiseerde applicatie. De Functioneel beheerder biedt ondersteuning en test de systemen uit

De **Applicatie Beheerder** is verantwoordelijk voor het onderhouden en beheren van de applicatie. Daarnaast zorgt deze persoon voor handleidingen en documentatie omtrent het gebruik en is verantwoordelijk voor eventuele installaties en configuraties van de applicatie

De **Technisch Beheerder** is verantwoordelijk voor de infrastructuur om de software heen. Denk hierbij aan servers om de applicatie online te laten draaien of andere aspecten. Ze doen niks met de Software zelf maar zorgen wel dat deze kan runnen.

# Beheertaken binnen DevOps

Er zijn diverse beheertaken binnen DevOps die kunnen worden ingedeeld in diverse catogorien Dit zijn: Ondersteuning, Onderhoud en Sturing

**Ondersteuning**

* Gebruikersondersteuning van de applicatie of Software
* Beheer van Gegevens van de applicatie of Software
* Autorisaties van gebruikers van de applicatie of Software
* Procedures of werkzaamheden van de applicatie of Software

De tak ondersteuning hoort bij de **Gebruikers**, dit kan zowel interne gebruikers als externe gebruikers zijn

**Onderhoud**

* Correctief, als er een fout is die gecorrigeerd moet worden.
* Preventief, om te voorkomen dat er een fout wordt gemaakt
* Perfectief, om de applicatie of software te perfectioneren.
* Adaptief, onderhoud dat wordt uitgevoerd op basis van verandering van buitenaf, zoals wet wijzigingen
* Additief, onderhoud omdat er een wijziging is in de functionele eisen van de applicatie of Software

Het onderhoud wordt uitgevoerd door **de Functioneel, Applicatie of Technisch Beheerder** op basis van hun functie en het type onderhoud.

**Sturing**

* Prestatieafspraken
* Rapportages
* Kosten
* Verbetering

De Sturing heeft te maken met de **kwaliteitsverbetering** en het monitoren hiervan

# Het Beheerproces binnen DevOps

Binnen DevOps zijn er diverse processen voor het beheer

Dit zijn:

**Request**

* Vraag of verzoek

**Incident**

* Storingen
* Foutmeldingen

**Problem**

* Structueel probleem

**Change**

* Wijzigingen

**Release**

* De Juiste en nieuwste versie

Deze processen hebben diverse triggers. Op basis van de Vragen en verzoeken en Incidenten bij de gebruikers ontstaan er problemen. De beheerders komen op basis van deze punten bij wijzigingen en nieuwe releases weer terug bij de gebruikers. Op deze manier blijft de cirkel van processen intact.

# Het belang van documentatie binnen DevOps

Documentatie maakt het makkelijker om te werken met de hard en software en helpt bij het maken van de juiste keuzes voor deze Software. Beslissingen die worden genomen zijn voor iedereen binnen handbereik i.p.v. dat het bij een iemand op de harde schijf staat. Door gebruik van documentatie vergroot je de kennis van het hele team.

# Wat is samenwerken binnen DevOps

Omdat er binnen DevOps diverse teams zijn die van elkaar afhankelijk zijn is samenwerken erg belangrijk om sneller en eenvoudiger de bedrijfsdoelen te behalen. Zo kan een Applicatiebeheerder zijn werk niet uitvoeren als het Technische beheer niet met ze samenwerkt. Alle teams moeten elkaar helpen en samenwerken om hun doelen te bereiken.

# Wat is Hardware?

Hardware vormt de fysieke basis van ICT-systemen en biedt de middelen en capaciteit voor gegevensverwerking, opslag, communicatie en interactie.

Kortom is er zonder Hardware geen ICT-Systeem. Hardware zorgt ervoor dat er gebruik kan worden van het apparaat. Er zijn vele vormen van hardware

Denk hierbij aan een processor, moederbord, videokaart/grafische kaart maar ook toetsenborden, muizen en beeldscherm is hardware.

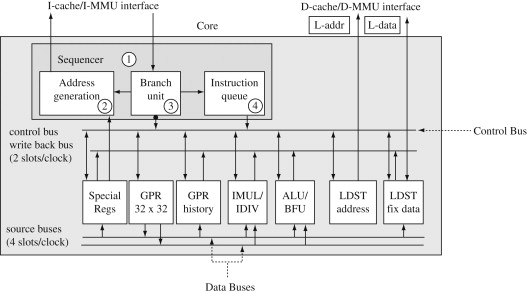
# De Algemene werking van Hardware binnen een computer?

Binnen een computer vinden we de volgende vormen hardware:

**Processor**

De processor wordt ook wel het hart van de computer genoemd. Deze hoofdchip is namelijk verantwoordelijk voor alle berekeningen en aansturingen die in een computer plaats vinden.

Hoe sneller een processor is, hoe meer berekeningen en aansturingen er per seconde kan verwerkt kan worden.



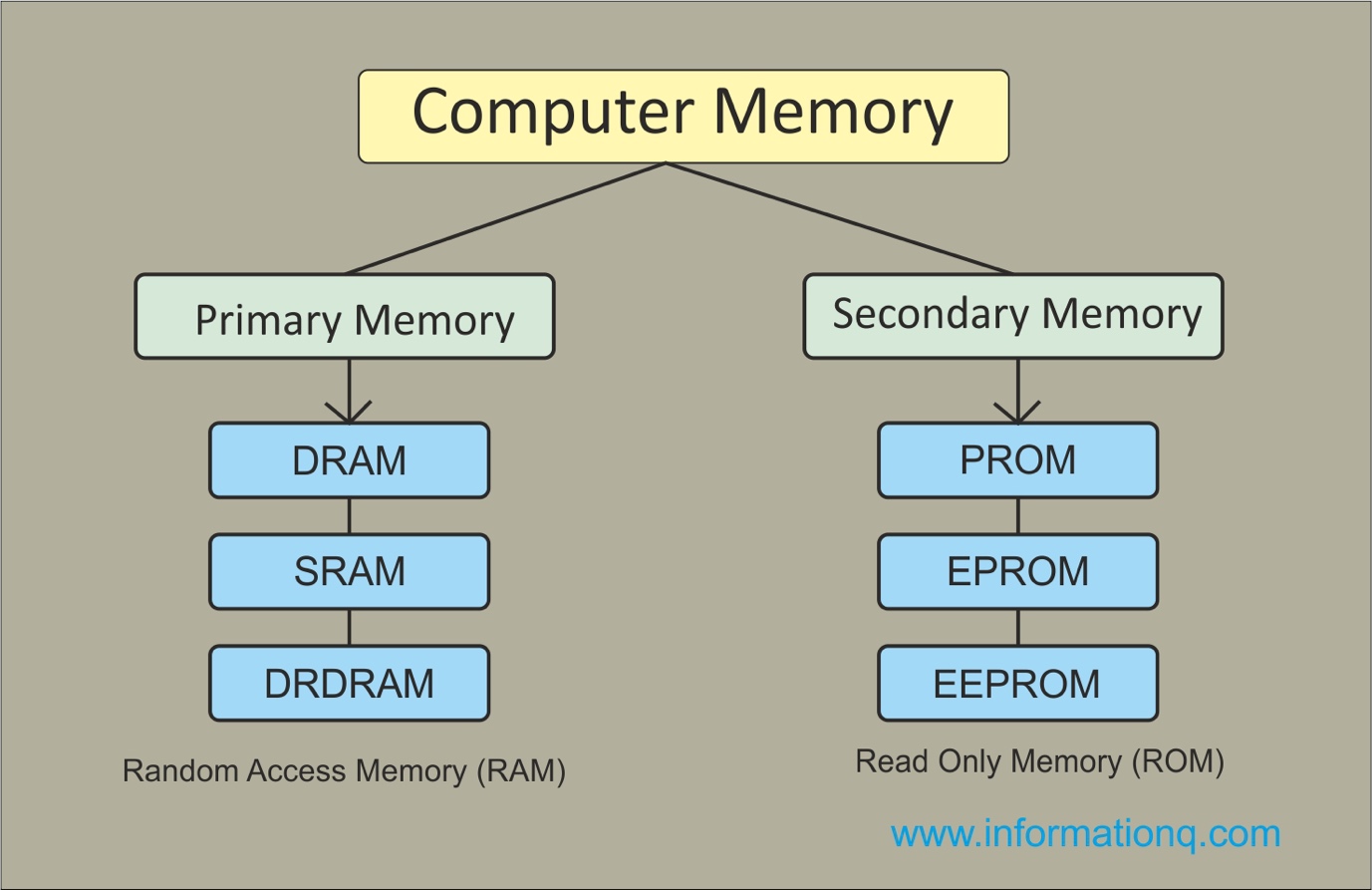
**Geheugen**

Geheugen, in het Engels computer memory wordt vaak uitgedrukt in RAM

RAM staat voor Random Access Memory en wordt ook wel werkgeheugen of intern geheugen genoemd. Dit werkgeheugen zorgt voor tijdelijke opslag van je besturingssysteem en de programma’s die je open hebt staan. Door RAM is het bijvoorbeeld mogelijk om snel tussen verschillende tabbladen in je browser te schakelen. Hoe meer RAM-geheugen je hebt in een laptop of desktop, des te meer processen je tegelijk uit kan voeren. Het interne geheugen werkt samen met de processor. Daarom werkt een computer met weinig RAM ook langzamer dan een pc met veel intern geheugen.

De snelheid van het RAM-werkgeheugen wordt aangegeven in MHz. Dit is de kloksnelheid van het interne geheugen en zegt iets over het aantal schakelingen per seconden dat het RAM-geheugen kan maken. Een hoger getal betekent dat het RAM-geheugen meer schakelingen per seconden uit kan voeren en dus sneller is.

Bij diverse samenstelling is het mogelijk om de kloksnelheid te verhogen. Dit proces heet “Overclocking”



**Harde schijf**

De Harde schijf, ook wel Hard Disk is verantwoordelijk voor de opslag binnen de computer

Over het algemeen bestaat harde schijven in twee vormen: harde schijven (HDD's) en solid state drives (SSD's).

Het meer 'traditionele' type harde schijf is HDD.

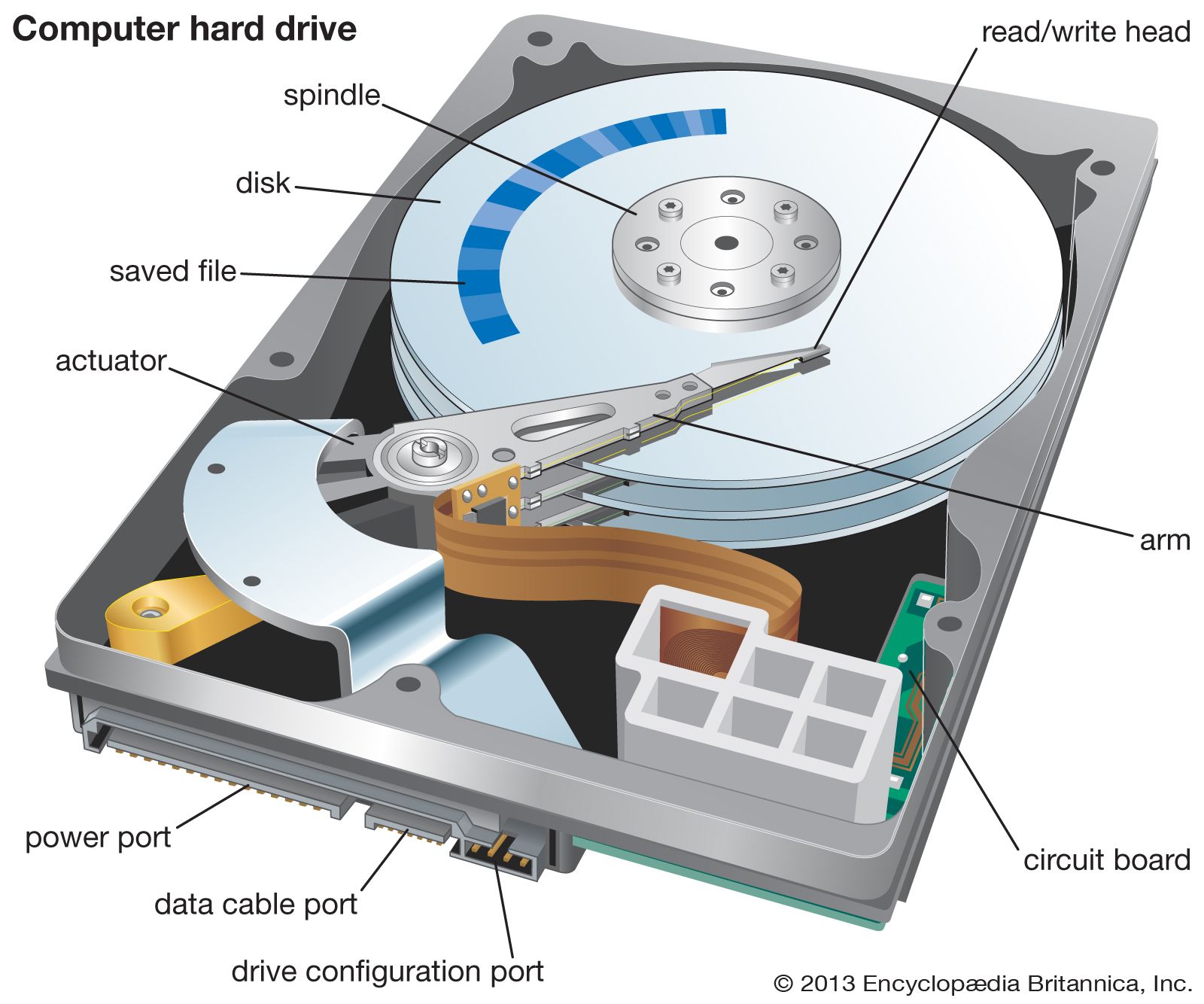
Harde schijven bestaan uit gemagnetiseerde schijven, ook wel platen genoemd, die snel ronddraaien, meestal ergens tussen de 5.400 en 15.000 RPM. Hoe sneller de magnetische schijf draait, hoe sneller je computer toegang heeft tot de informatie die erop staat.

Tegenwoordig kunnen interne HDD's een maximale capaciteit van 20 TB bereiken. Sinds de opkomst van SSD worden harde schijven zelden gebruikt als secundaire opslag van een computer, maar zijn ze nog steeds betrouwbaar als externe opslagoptie.

SSD's (solid state drives) zijn het nieuwere type harde schijf. Ze krijgen de voorkeur voor gebruik als interne harde schijf in high-end laptops, en alle smartphones en tablets gebruiken ook een vorm van SSD.

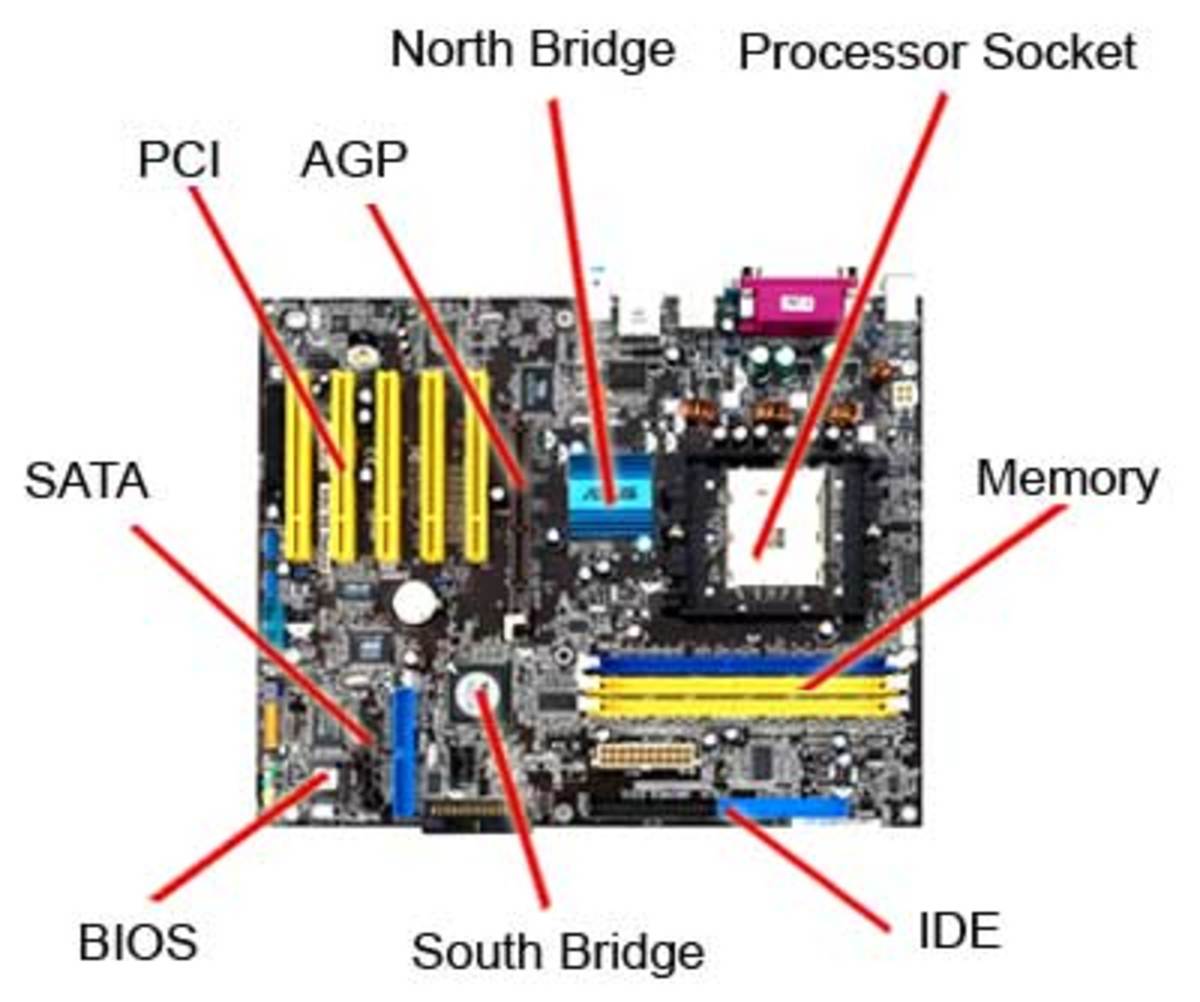
SDD's zijn veel duurder om te produceren, dus hoewel ze steeds vaker voorkomen als de primaire schijf voor high-end laptops en pc's, genieten harde schijven nog steeds vaak de voorkeur als een goedkopere externe optie.

Voor High end computers wordt vaak de SDD gebruikt om het besturing systeem op te runnen en de HDD voor overige opslag. Op deze manier blijft de computer namelijk snel



**Moederbord**

Het moederbord is een printplaat waarop alle overige onderdelen worden aangesloten. Het moederbord zorgt ervoor dat alles met elkaar kan samenwerken en communiceren.



**Grafische kaart/VideoKaart**

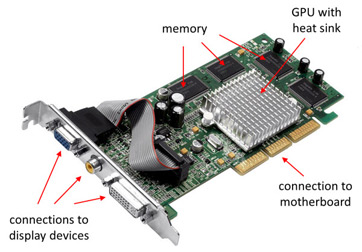
Een grafische kaart –of videokaart– zou u als een processor voor de grafische prestaties van uw computer kunnen beschouwen. Het hart van de videokaart is de GPU, wat staat voor Graphics Processing Unit. De GPU berekent de beeldopbouw en versnelt de weergave van 3D-beelden. Grafische kaarten zijn dan ook vooral gewild bij computergebruikers die graag games spelen.

Tegenwoordig zit er vaak in de CPU (processor) een videochip ingebouwd.

Dit is echter niet genoeg voor de “zwaardere games”

De grootse fabrikanten van videokaarten zijn NVIDIA en AMD.

Echter zijn er tegenwoordig ook nieuwe aanbieders zoals ASUS.



**Koeler**

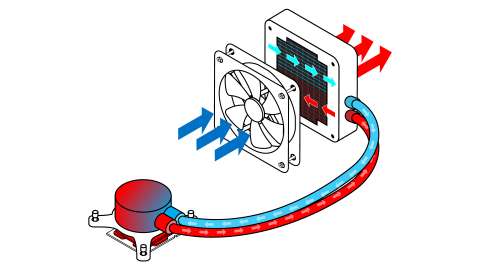
Een veel voorkomend onderdeel binnen de moderne pc’s in de koeler

Waarom is koelen belangrijk? Koelen is belangrijk voor zowel de levensduur als de prestaties van je componenten. Tegenwoordig komt zware directe schade door oververhitting eigenlijk niet meer voor, omdat je processor en je gpu hun snelheid automatisch remmen als de temperatuur te hoog wordt. Dat heet “Thermal Throttling”, en dat wil je natuurlijk voorkomen. Je haalt dan niet het maximale uit je hardware. Op welke temperatuur dit gebeurt is afhankelijk van het model GPU of CPU dat je hebt, daar is geen vaste waarde voor. Ondanks die ingebouwde beveiliging is het nog steeds beter voor je componenten om zo koel mogelijk te blijven, om de levensduur en prestaties van je componenten te maximaliseren.

Er zijn hier 2 verschillende vormen van, de liquid (Water) en de Air (lucht)

Een luchtkoeler gebruikt een combinatie van heatsinks - en vaak ook heatpipes - om de door de processor opgewekte warmte op te nemen waarna hij door één of twee ventilatoren wordt weggeblazen.De computer gerelateerde waterkoelsystemen bestaan vaak ​​uit een waterblok, een pomp, een radiator, de leidingen en optioneel een extra reservoir. De pomp blaast de vloeistof door het computersysteem. De verplaatste vloeistof stroomt door het waterblok welke bevestigd is aan de CPU of GPU. Hier wordt de warmte van het onderdeel naar de vloeistof overgebracht. Een enkel systeem kan één of meerdere waterblokken hebben.

Het door de computer opgewarmde water gaat naar een radiator waar ventilatoren koele lucht over de leidingen blazen en koelventilatoren om de warmte af te voeren. Een reservoir kan extra water bevatten om hierdoor meer thermische massa en watercapaciteit te organiseren. Flexibele of harde leidingen of buizen transporteren het water door het systeem. Vaak wordt er een aangroeiwerende vloeistof aan het water toegevoegd om te voorkomen dat bacteriën of algen in de computer gaan groeien, wat werkt als een soort water filter.



# Welke hardware zit minimaal in een computer op programma’s te kunnen runnen?

Om een programma te kunnen runnen zijn er systeemeisen aan verbonden. Een bepaald programma kan namelijk meer eisen hebben dan een ander.

De eisen van Windows 10 zijn bijvoorbeeld als volgt:

* Processor: 1 gigahertz (GHz) of sneller
* Intern geheugen (RAM): 2 gigabyte (GB)
* Harde schijf: 16 tot 20 GB vrije ruimte
* Grafische kaart: DirectX 9 of nieuwer met WDDM 1.0 stuurprogramma
* Beeldscherm: 800 x 600

De minimale requirements verschillen per software, zo zal een zware game voor meer geheugen nodig hebben en een betere grafische kaart dan Windows 10 nodig heeft.

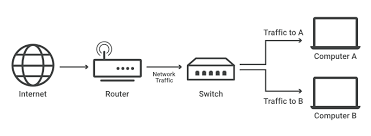
# Hoe zit een netwerk in elkaar?

Een netwerk heeft een fysieke en logica kant.

De Fysieke kant bestaat uit de volgende onderdelen:

* **Netwerk kabel**
* **Hub**
* **Switch**
* **Router**
* **Firewall**
* **Gateway**

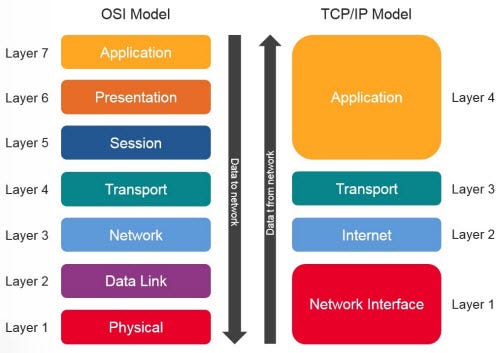
Dit kan bijvoorbeeld uitzien als het volgende schema:



Naast de fysieke kant is er ook nog de logica kant

De logica kant bestaat uit de volgende onderdelen

* **OSI Layers**
* **TCP Layers**
* **IP Layers**



# De belangrijkste protocollen voor netwerken staan beschreven (minimaal 4)

Internet Protocol (IP) is de belangrijkste set (of communicatieprotocol) van digitale berichtenformats en regels voor het uitwisselen van berichten tussen computers over een enkel netwerk of een reeks onderling verbonden netwerken, met behulp van de Internet Protocol Suite (vaak TCP/IP genoemd).

De belangrijkste en meest voorkomende protocollen zijn als volgt:

**RFC-786 – User Datagram Protocol (UDP)**

**RFC-791 – Internet Protocol (IP)**

**RFC-793 - Transmission Control Protocol (TCP)**

**RFC-1945 – Hypertext Transfer Protocol (HTTP)**

Dit zijn de belangrijkste en meest voorkomende protocollen

# De netwerkactiviteiten van een verbinding tussen twee computers staat beschreven

**Verbinding tot stand brengen:**

De twee computers, die elk zijn voorzien van een netwerkinterface (bijvoorbeeld een Ethernet-kaart of een draadloze adapter), zijn fysiek of via een netwerk verbonden.

De computers kunnen zich in hetzelfde lokale netwerk (LAN) bevinden of via internet met elkaar worden verbonden.

**IP-adresconfiguratie:**

Elke computer heeft een uniek IP-adres toegewezen om deze te identificeren in het netwerk.

Als de computers zich in verschillende netwerken bevinden, wordt mogelijk gebruik gemaakt van NAT (Network Address Translation) om de interne IP-adressen om te zetten naar een extern IP-adres.

**Protocolselectie:**

De computers moeten overeenkomen welk communicatieprotocol ze zullen gebruiken, zoals TCP of UDP.

**Poortselectie:**

Applicaties op beide computers luisteren op bepaalde poorten voor inkomende communicatie.

De zender moet de juiste poort op de ontvanger bereiken om de gewenste applicatie te bereiken.

**Verbindingsverzoek:**

De zender (meestal de client) stuurt een verzoek naar de ontvanger (meestal de server) om een verbinding tot stand te brengen. Dit kan bijvoorbeeld een HTTP-verzoek zijn als de computers via het web communiceren.

**Verbindingsopbouw:**

De ontvanger accepteert het verbindingsverzoek en er wordt een verbinding tot stand gebracht. Na het opbouwen van de verbinding kunnen gegevens worden verzonden.

**Gegevensoverdracht:**

Gegevens worden in pakketten verdeeld en over de verbinding verzonden.

Er kunnen verschillende protocollen en technologieën worden gebruikt om ervoor te zorgen dat de gegevens veilig, betrouwbaar en in de juiste volgorde worden afgeleverd.

**Gegevensontvangst:**

De ontvanger ontvangt de gegevenspakketten, voert eventuele controles uit en zorgt voor assemblage in de juiste volgorde.

De gegevens worden doorgegeven aan de applicatie die ze kan verwerken.

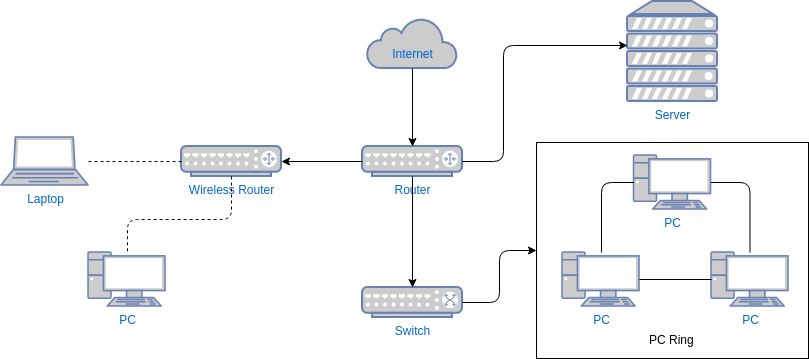
**Verbindingsafsluiting:**

Na het voltooien van de communicatie worden de computers ontkoppeld.

**Beveiliging:**

Afhankelijk van de vereisten van de communicatie kunnen versleuteling en beveiligingsprotocollen zoals SSL/TLS worden toegepast.

# Een netwerkdiagram tussen eigen computer en server is gegeven



# Er staat beschreven wat Docker is

Docker is software die gemaakt is voor systeembeheerders en ontwikkelaars. Het kan vooral nuttig zijn voor toepassingen zoals serverdiensten en voor commandline-tools. In theorie krijg je ook grafische applicaties aan de praat met Docker, maar het is niet de eerste keuze voor desktopsoftware.

Docker gebruikt containertechnologie: dat wordt vaak op één hoop gegooid met virtualisatie, maar verschilt daar fundamenteel in. Een container is geen virtuele ma­chine, maar een normaal proces dat wordt wijsgemaakt dat het zich in een optimale nepwereld bevindt.

Binnen Docker kun je dus geen compleet besturingssysteem installeren, zoals in een virtuele machine. In een Docker-container kan bijvoorbeeld serversoftware draaien zoals de Nginx webserver, Pi-hole adblocker of MariaDB database.

Het makkelijkst gebruik je containers met Linux als basis, op een computer (host) die ook op Linux draait. Met Windows 10 of hoger is het ook mogelijk Linux-containers op Windows te draaien (dankzij Windows Subsystem for Linux versie 2, WSL2). Windows-containers op Windows-servers zijn een verhaal op zich.

## Docker VS Virtual Machine

* Beide zijn geïsoleerd.
* Docker draait native en heeft bijna geen performance impact.
* Op Windows draait Docker vaak onder WSL2, WSL2 is wel een virtual machine.
* Docker containers starten snel en zijn flexibel, daarom veel gebruikt voor CI/CD (DevOps).

# De procedure om een Dockeromgeving in eigen context te maken staat beschreven

Voor de installatie van Docker zijn er 2 mogelijkheden. De client kan via de website van Docker worden gedownload wat wij hebben gedaan. Ook zijn er via PHPstorm mogelijkheden tot plugins. Na de installatie van Docker openen wij PHPstorm en ons project. Daarna gaan wij naar de instellingen doormiddel van de cmd en 8 toets (macbook shortcut)

Daar voegen wij Docker toe aan de services van PHPstorm

Na een kleine test door op apply te drukken kan er worden gekeken of er verbinding wordt gemaakt. Als dit succesvol is koppelen wij de docker aan phpstorm

Als dit is gedaan maken wij een bestand aan met de naam dockerfile

De dockerfile bevat de voglende code:

FROM php:8.2-apache

COPY . /var/www/html

EXPOSE 80

Deze runnen wij en dit maakt automatisch een docker container aan. Als daarna de localhost vanuit wordt gestart kan er worden gezien in de dockerclient dat de docker werkt.

# Een eigen Dockeromgeving staat beschreven

Wij hebben de vorige stappen uitgevoerd voor ons Laravel PHP project.

Op deze manier hebben wij een container aangemaakt die ons project runt.

# Foto's van eigen Dockeromgeving worden in draaiende staat getoond

Afbeelding met schermopname, tekst, software, Computerpictogram

Automatisch gegenereerde beschrijving

# Er staat beschreven wat acceptatiecriteria zijn

Acceptatiecriteria is eigenlijk de criteria, dus wat nodig is voor een project om acceptabel te zijn. Dit zijn dus eisen voor een project waar het altijd aan moet voldoen. Vaak heeft een klant een heel scherp idee wat er nodig is. Maar over de details wordt vaak niet voldoende nagedacht.

# Van een eigen project staan minimaal vijf acceptatiecriteria beschreven

Voor het project bij Quasir waren de volgende criteria van belang;

* Webapplicatie moest mobile ready zijn
* Klanten moesten melding kunnen maken
* De Website moet snel laden
* Er moet een knop zijn om een mail te kunnen sturen zodat registeren niet verplicht is
* Overzichtelijke website met een design dat aansluit op de huidige website van Quasir

# Er staan meerdere manieren van testen beschreven

De volgende manieren van Testen zijn er:

* Functionele testen, waarbij je checkt of de ontwikkelde functionaliteiten doen wat ze zouden moeten doen;
* Niet-functionele testen, waarbij je op niet functionele zaken zoals gebruiksvriendelijkheid checkt of de webapplicatie naar behoren werkt;

Bij functionele testen denken wij dan aan:

* Unittest. Hierbij schrijft de ontwikkelaar vooraf een geautomatiseerde test waarmee een deel van de code van een webapplicatie wordt getest op het te verwachten resultaat. Komt er een ander resultaat uit dan vooraf door de ontwikkelaar gedefinieerd? Dan gaat er iets fout in de applicatie.
* Smoketest. Een test die aangeeft of de basisfunctionaliteiten werken, de zogenoemde
* happy flow. Is dat inderdaad het geval, dan kan de software verder ontwikkeld en getest worden. Dit is vaak een snelle eerste test die door iemand anders dan de ontwikkelaar wordt uitgevoerd.
* Regressietest. Met een regressietest check je of de systeemonderdelen waarvoor de software níet is aangepast nog werken nadat er nieuwe functionaliteiten zijn toegevoegd.
* Integratietest. Hiermee controleer je of verschillende onderdelen van de software goed samenwerken.

Bij niet functionele testen denken wij aan:

* Gebruikerstest. Hoe gebruiksvriendelijk is de webapplicatie nadat de nieuwe software is geïmplementeerd? In een gebruikerstest laat je een groep gebruikers alle functies van de webapplicatie testen, liefst op verschillende browsers en apparaten. Waar lopen de gebruikers tegenaan, kunnen ze alles eenvoudig vinden en hoe kunnen deze fouten/ongemakken worden verholpen?
* A/B-testen. Door middel van een A/B-test kun je bijvoorbeeld achterhalen of de website in een bepaalde weergave (bijvoorbeeld kleurstelling, video, lay-out of CTA-button) tot meer conversie leidt. De ene helft van de gebruikers krijgt versie A te zien, de andere helft versie B. Welke scoort beter?

# Op basis van een eigen project is per acceptatiecriterium een test uitgewerkt (inclusief code)

1. **Webapplicatie moest mobile ready zijn:**

**<?php**

**namespace Tests\Browser;**

**use Laravel\Dusk\Browser;**

**use Tests\DuskTestCase;**

**class MobileReadinessTest extends DuskTestCase**

**{**

**/\*\***

**\* A basic mobile readiness test example.**

**\***

**\* @return void**

**\*/**

**public function testMobileReadiness()**

**{**

**$this->browse(function (Browser $browser) {**

**$browser->resize(360, 640); // Set the browser window size to emulate a mobile device**

**$browser->visit('/'); // Change '/'' to the URL of your homepage**

**});**

**}**

**}**

1. **Klanten moesten melding kunnen maken:**

**<?php**

**namespace Tests\Browser;**

**use Illuminate\Foundation\Testing\DatabaseMigrations;**

**use Laravel\Dusk\Browser;**

**use Tests\DuskTestCase;**

**class ForumTest extends DuskTestCase**

**{**

**use DatabaseMigrations;**

**/\*\***

**\* Test creating a forum post.**

**\***

**\* @return void**

**\*/**

**public function TestCreateFormPost()**

**{**

**$this->browse(function (Browser $browser) {**

**$browser->visit('/ticket')**

**->type('title', 'Dit is de titel van mijn bericht')**

**->type('message', 'Dit is de inhoud van mijn bericht')**

**->press('Versturen') // Update button label**

**->assertPathIs('/forum'); //**

**});**

**}**

**}**

1. **De Website moet snel laden:**

**PASS Tests\Browser\HomePageLoadTest**

**✓ home page load time**

**namespace Tests\Browser;  
  
use Illuminate\Foundation\Testing\DatabaseMigrations;  
use Laravel\Dusk\Browser;  
use Tests\DuskTestCase;  
  
class HomePageLoadTest extends DuskTestCase  
{  
 /\*\*  
 \* A basic browser test example.  
 \*  
 \* @return void  
 \*/  
 public function testHomePageLoadTime()  
 {  
 $this->browse(function (Browser $browser) {  
 $start = microtime(true); // Record the start time  
 $browser->visit('/'); // Change '/'' to the URL of your homepage  
 $end = microtime(true); // Record the end time  
  
 $loadTime = $end - $start;  
 $this->assertTrue($loadTime < 5, "Homepage loaded in $loadTime seconds"); // Adjust the threshold as needed  
 });  
 }  
}**

1. **Er moet een knop zijn om een mail te kunnen sturen zodat registeren niet verplicht is: <?php**

**namespace Tests\Browser;**

**use Illuminate\Foundation\Testing\DatabaseMigrations;**

**use Laravel\Dusk\Browser;**

**use Tests\DuskTestCase;**

**class TestClickAnchorLink extends DuskTestCase**

**{**

**/\*\***

**\* A Dusk test example.**

**\*/**

**public function testClickAnchorLink()**

**{**

**$this->browse(function (Browser $browser) {**

**// Visit the root URL.**

**$browser->visit('/')**

**->clickLink('Klik hier');**

**});**

**}**

**}**

1. **Beveiligde admin page:**

**Tests\Browser\AdminDashboardAccessTest > unauthorized user cannot access admin dashboard.**

**<?php**

**namespace Tests\Browser**

**use Illuminate\Foundation\Testing\DatabaseMigrations.**

**use Laravel\Dusk\Browser.**

**use Tests\DuskTestCase;**

**class AdminDashboardAccessTest extends DuskTestCase**

**{**

**use DatabaseMigrations;**

**/\*\***

**\* Test unauthorized access to the admin dashboard.**

**\***

**\* @return void**

**\*/**

**public function testUnauthorizedUserCannotAccessAdminDashboard()**

**{**

**$this->browse(function (Browser $browser) {**

**// Visit the admin dashboard URL.**

**$browser->visit('/admin/')**

**->assertSee('/login'); // Assumes that unauthorized users are redirected to the login page.**

**});**

**}**

**}**

# Er staat beschreven waarom testen belangrijk zijn

Zonder testen is het niet duidelijk of de software werkt zoals die hoort te werken.

Doormiddel van testscripts en handmatig testwerk kan er worden gekeken of de software ook doet waarvoor het bedoeld is, eventuele functies werken naar behoren en of er gaat fouten in de software of de security van de software zit. Op deze manier ervaart de klant zo min mogelijk problemen.

# Er zijn minimaal 5 unit tests uitgevoerd op units met een mogelijke return waarde. Deze unittests moeten de goede werking van de units aantonen.

Unit test 1: Is de Admin omgeving afgeschermd voor overige gebruikers.

Eindresultaat Unit Test 1:

Voor deze test hebben wij gebruik gemaakt van Laravel Dusk;

Console gaf de volgende waarde terug;

**Tests\Browser\AdminDashboardAccessTest > unauthorized user cannot access admin dashboard.**

De source code van de test is als volgt:

**<?php**

**namespace Tests\Browser**

**use Illuminate\Foundation\Testing\DatabaseMigrations.**

**use Laravel\Dusk\Browser.**

**use Tests\DuskTestCase;**

**class AdminDashboardAccessTest extends DuskTestCase**

**{**

**use DatabaseMigrations;**

**/\*\***

**\* Test unauthorized access to the admin dashboard.**

**\***

**\* @return void**

**\*/**

**public function testUnauthorizedUserCannotAccessAdminDashboard()**

**{**

**$this->browse(function (Browser $browser) {**

**// Visit the admin dashboard URL.**

**$browser->visit('/admin/')**

**->assertSee('/login'); // Assumes that unauthorized users are redirected to the login page.**

**});**

**}**

**}**

Unit test 2:

Werkt de email knop, zodat gebruikers ook zonder account een melding kunnen maken.

Eindresultaat Unit Test 2: Voor deze test hebben wij gebruik gemaakt van Laravel Dusk;

De source code van deze test:

**<?php**

**namespace Tests\Browser;**

**use Illuminate\Foundation\Testing\DatabaseMigrations;**

**use Laravel\Dusk\Browser;**

**use Tests\DuskTestCase;**

**class TestClickAnchorLink extends DuskTestCase**

**{**

**/\*\***

**\* A Dusk test example.**

**\*/**

**public function testClickAnchorLink()**

**{**

**$this->browse(function (Browser $browser) {**

**// Visit the root URL.**

**$browser->visit('/')**

**->clickLink('Klik hier');**

**});**

**}**

**}**

Unit test 3: Werkt het maken van een Ticket;

Eindresultaat Unit Test 3: Voor deze test hebben wij gebruik gemaakt van Laravel Dusk;

Als waarde kregen wij het volgende terug:

Tests\Browser\ForumTest

create forum post

Hierbij werd de volgende source code gebruikt;

**<?php**

**namespace Tests\Browser;**

**use Illuminate\Foundation\Testing\DatabaseMigrations;**

**use Laravel\Dusk\Browser;**

**use Tests\DuskTestCase;**

**class ForumTest extends DuskTestCase**

**{**

**use DatabaseMigrations;**

**/\*\***

**\* Test creating a forum post.**

**\***

**\* @return void**

**\*/**

**public function TestCreateFormPost()**

**{**

**$this->browse(function (Browser $browser) {**

**$browser->visit('/ticket')**

**->type('title', 'Dit is de titel van mijn bericht')**

**->type('message', 'Dit is de inhoud van mijn bericht')**

**->press('Versturen') // Update button label**

**->assertPathIs('/forum'); //**

**});**

**}**

**}**

Unit 4 Werkt het in en uitklappen van de Q&A op de hoofdpagina

**class HomepageTest extends DuskTestCase  
{  
  
 public function testFAQPage()  
 {  
 $this->browse(function (Browser $browser) {  
 $browser->visit('/') // Replace '/' with the URL of your homepage  
 ->assertSee('Over Quasir')  
 ->assertSee('FAQ')  
 ->assertVisible('.qa\_\_question')  
 ->assertMissing('.qa\_\_answer');  
  
 // Simulate clicking the "button" to open the "qa\_container"  
 $browser->click('.qa\_\_question img.arrow'); // Add 'arrow' to the selector  
  
 // Now, click on the image with id 'profileLogin' to go to the login page  
 $browser->click('#profileLogin');  
  
 // Add any additional assertions or interactions related to the login page here  
 });  
 }  
}**

Unit 5: Testen of er een acocunt kan worden aangemaakt en de user daarna wordt geredirect naar hun persoonlijk dashboard.

**namespace Tests\Browser;  
  
use Laravel\Dusk\Browser;  
use Tests\DuskTestCase;  
  
class RegisterTest extends DuskTestCase  
{  
 /\*\*  
 \* Test user registration.  
 \*  
 \* @return void  
 \*/  
 public function testUserRegistration()  
 {  
 $this->browse(function (Browser $browser) {  
 $browser->visit('/register') // Replace with the URL of your registration page  
 ->type('firstname', 'John') // Fill in the registration form fields  
 ->type('lastname', 'Doe')  
 ->type('phone', '0624976011')  
 ->type('email', 'johndoe@example.com')  
 ->type('password', 'password')  
 ->press('Register') // Submit the registration form  
  
 // Add assertions for a successful registration, e.g., redirection or success message  
 ->assertPathIs('/dashboard'); // Replace with the URL where users are redirected after successful registration  
 });  
 }  
}**

# Er zijn end-to-end test beschreven en uitgevoerd

Voor de End-to-end test hebben wij een test geschreven voor het hele authenticatie stuk, van login en registratie naar het dashboard voor de Users:

**<?php  
  
namespace Tests\Browser;  
  
use Illuminate\Foundation\Testing\DatabaseMigrations;  
use Laravel\Dusk\Browser;  
use Tests\DuskTestCase;  
  
class EndToEndTestRegister extends DuskTestCase  
{  
 /\*\*  
 \* A basic end-to-end test example.  
 \*  
 \* @return void  
 \*/  
 public function testEndToEndScenario()  
 {  
 $this->browse(function (Browser $browser) {  
 // Register a new user  
 $browser->visit('/register'); // Change '/register' to your registration page URL  
 $browser->type('firstname', 'John');  
 $browser->type('lastname', 'Doe');  
 $browser->type('phone', '0612345678');  
 $browser->type('email', 'johndoe@example.com');  
 $browser->type('password', 'password');  
 $browser->press('Register'); // Change to the button's text or element to submit the form  
  
 $browser->visit('/login'); // Change '/login' to your login page URL  
 $browser->type('email', 'johndoe@example.com');  
 $browser->type('password', 'password');  
 $browser->press('Login'); // Change to the button's text or element to submit the login form  
  
 // Check if the user is authenticated (logged in)  
 $browser->assertAuthenticated();  
  
 // If the user is authenticated, proceed to the dashboard or perform other actions  
 $browser->visit('/dashboard'); // Change '/dashboard' to the user dashboard URL  
  
 // Perform actions on the logged-in user, e.g., navigate around the application  
  
 // Log out**

**$browser->click('.logout); //   
 // Check if the user is successfully logged out by verifying that they are on the login page  
 $browser->assertPathIs('/login'); // });  
 }  
}**

# In het portfolio wordt in eigen woorden beschreven wat de voor- en nadelen van een Devopspipeline zijn

## Wat is een DevOps – pipeline:

Een DevOps-pipeline is een reeks geautomatiseerde processen en tools waarmee ontwikkelaars kunnen samenwerken aan het bouwen en implementeren van code in een productieomgeving

## Voordelen van de DevOps- Pipeline:

Het grootse voordeel van Automatisering binnen de DevOps-pipeline is dat dit heel veel tijd scheelt. Vooral als een bedrijf diverse dingen vaker maakt kunnen ook de testscript worden hergebruikt. Ook het deployen van de software gaat sneller want ook dit is geautomatiseerd.

## Nadelen van de DevOps-Pipeline:

Het grootste nadeel wat er bij het algemene begrip van automatisering is dat niet alles kan worden gezien door een testscript. Wat wij meemaken op onze stage is dat na het uitvoeren van testscript er 2 collega’s gaan proberen om de software te “slopen”. Vaak komen ze dan nog achter dingen waar de testscript geen problemen ondervonden. Het nadeel van zo automatische pipeline met deployment is dus dan ook dat er foutgevoelig software doorheen kan komen.

# In het portfolio wordt in eigen woorden beschreven wat CI-CD inhoudt, onderbouwd met minimaal 3 eigen praktijkvoorbeelden

CI/CD staat voor Continuous Integration en Continuous Delivery, oftewel het process om de cyclus van software development te automatiseren. De Processen zorgen ervoor dat aspecten zoals Testen en deployment en building van de software automatisch gaat.

# In het portfolio is een zelfgeschreven CI-bestand toegevoegd

Ons ***.gitlab-ci.yml*** bestand ziet eruit als volgt:

**stages:**

**- build**

**- test**

**- deploy**

**variables:**

**# Define your environment-specific variables here**

**APP\_ENV: Quasir**

**DB\_CONNECTION: mysql**

**DB\_DATABASE: Quasir**

**before\_script:**

**- cp .env.example .env**

**- composer install --no-interaction**

**- php artisan key:generate**

**- php artisan migrate**

**cache:**

**paths:**

**- vendor/**

**- node\_modules/**

**build:**

**stage: build**

**script:**

**- npm install**

**- npm run production**

**test:**

**stage: test**

**script:**

**- vendor/bin/phpunit**

**- php artisan dusk**

**deploy:**

**stage: deploy**

**script:**

**- echo "Deploy Quasir Applicatie"**

# Er is een uitgewerkte GIt omgeving opgezet conform het geleerde format

De Link naar onze git omgeving:

**https://github.com/Ruben168k/Quasir**

# Het archief met changes en versies toegevoegd van de Git omgeving

Afbeelding met tekst, schermopname, software, Computerpictogram

Automatisch gegenereerde beschrijving