



# Sofware Design Essentials

Johan van den Broek

# Inzicht in systeemontwerppatronen

# Architecturale modellen



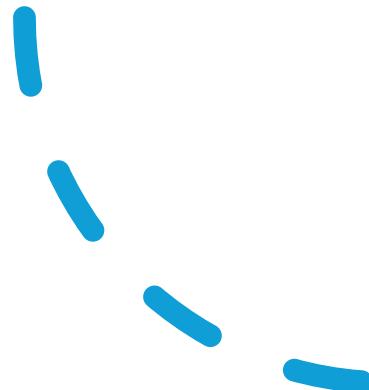
# Architecturale modellen

- Architecturale modellen worden gebruikt om de **structuur** en het ontwerp van een softwareapplicatie te definiëren en te organiseren.
- Deze modellen bieden een raamwerk waarbinnen verschillende componenten van de software en hun interacties worden beschreven.
- Architecturale modellen bieden een **blauwdruk** voor het ontwerpen en organiseren van de verschillende **componenten** van een softwareapplicatie.
- Dit helpt ontwikkelaars en andere stakeholders om het systeem beter te begrijpen, te bouwen, en te onderhouden.



# Typische Architecturale Modellen

- Monolithische architectuur
- Client-server architectuur
- Gelaagde architectuur (Layered Architecture or n-tier architecture)
- Service-georiënteerde architectuur (Service-Oriented Architecture)



# Monolithische architectuur

Architecturale modellen

# Monolithische architectuur

---

De term "monolithische architectuur" verwijst naar een softwareontwerppatroon waarin alle componenten van een applicatie als één samenhangende eenheid zijn geïntegreerd.

---

Dit betekent dat de applicatie bestaat uit één enkele, autonome codebasis die alle zakelijke functies omvat, die typisch als één groot project worden ontwikkeld en onderhouden.

---

De monolithische aanpak was de standaardmanier van softwareontwikkeling voordat de meer gedecentraliseerde benaderingen zoals werken met services populair werden.

# **Kenmerken van monolithische architectuur**

- Geïntegreerde codebasis**

- Alle code voor de applicatie, inclusief front-end en back-end componenten, databasetoegang, en bedrijfslogica, bevindt zich in één enkel project of oplossing. Dit kan het bouwen, testen en deployen initieel eenvoudiger maken omdat alles in één plaats is.

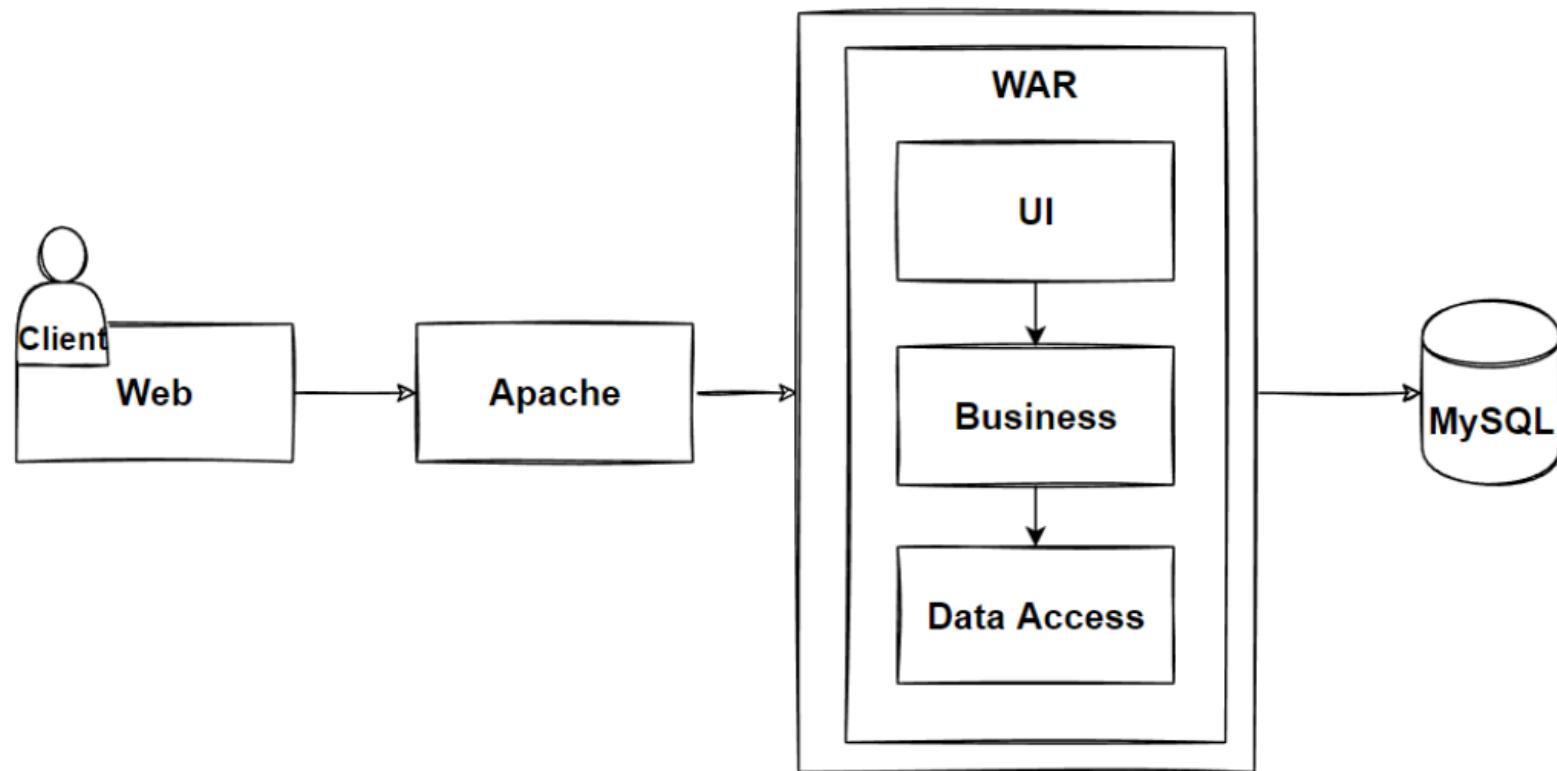
- Uniforme ontwikkelomgeving**

- Ontwikkelaars werken in een uniforme ontwikkelomgeving en gebruiken dezelfde frameworks en bibliotheken voor alle aspecten van de applicatie. Dit kan leiden tot minder configuratiecomplexiteit tijdens de ontwikkelingsfase.

- Eenvoudige deployment**

- Deployment is relatief rechtlijnig omdat het slechts het uitrollen van één enkele uitvoerbare file of directory omvat. Dit kan deploymentprocessen in minder complexe applicaties versnellen.

# Monolithische architectuur



Afbeelding van <https://medium.com/design-microservices-architecture-with-patterns/when-to-use-monolithic-architecture-57c0653e245e>

# Voordelen

---

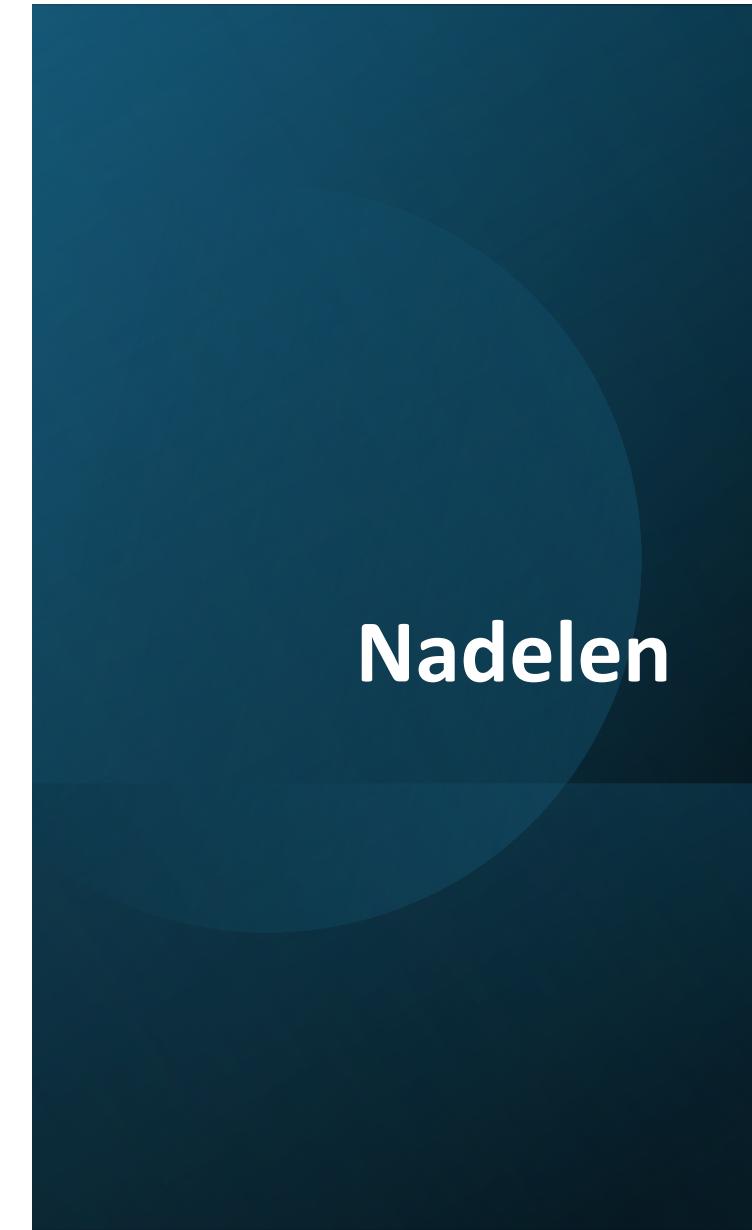
**Eenvoud in beheer:** Een enkele codebasis en applicatie om te beheren kan logistiek eenvoudiger zijn in kleinere of minder complexe projecten.

---

**Minder overhead:** Geen netwerkvertragingen of data-integriteit issues die kunnen voorkomen met gedistribueerde systemen.

---

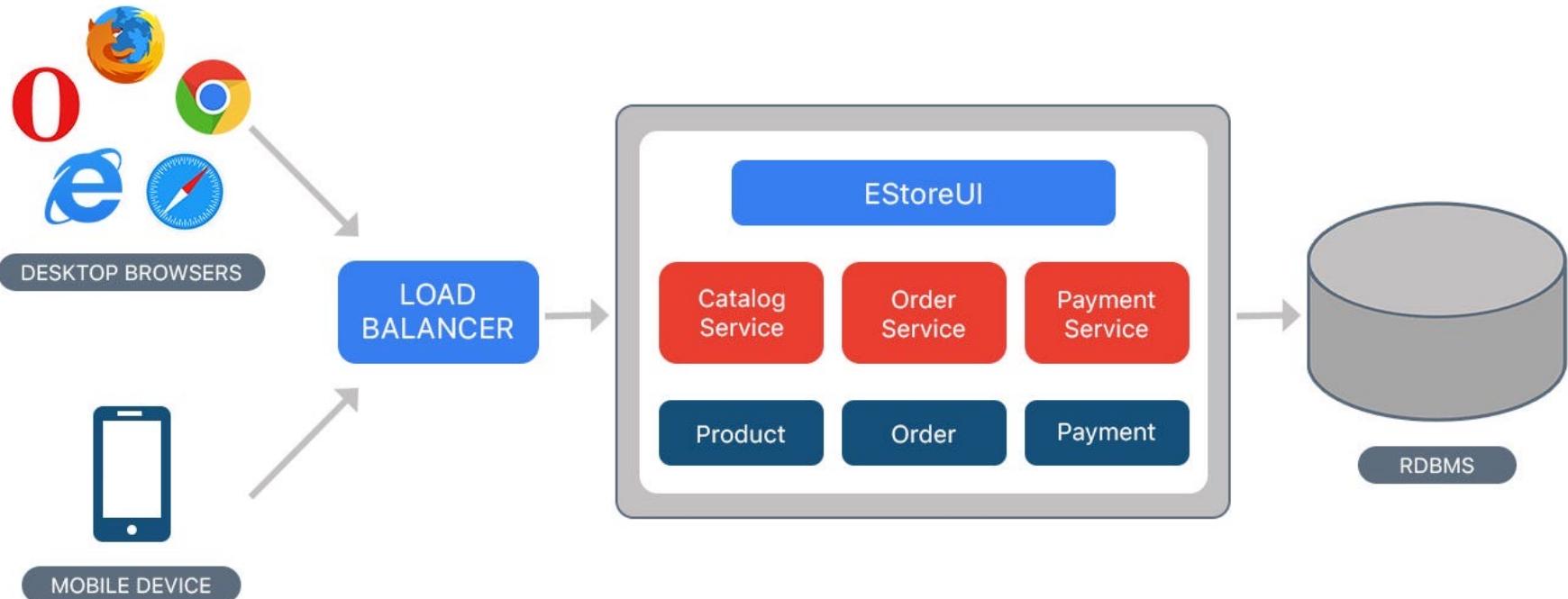
**Vereenvoudigde ontwikkeling en testing:** In de beginfases kan het eenvoudiger zijn om nieuwe functies te integreren en te testen wanneer alles deel uitmaakt van dezelfde applicatie.



## Nadelen

- **Schaalbaarheidsproblemen:** Als de applicatie groeit, kan het steeds moeilijker worden om de gehele applicatie effectief te schalen, aangezien kleinere updates de hele applicatie kunnen beïnvloeden.
- **Moeilijk bij onderhoud:** Grottere codebases kunnen onoverzichtelijk en moeilijk te onderhouden worden, vooral als verschillende functies sterk met elkaar verweven zijn.
- **Verhoogd risico op downtime:** Een fout in een klein deel van de applicatie kan de hele applicatie offline halen, wat bij gedecentraliseerde benaderingen minder waarschijnlijk is.
- **Tragere deployments:** Naarmate de applicatie groeit, kunnen deployments trager worden omdat elke kleine verandering een volledige hercompilatie en deploy van de hele applicatie vereist.

# Monolithische architectuur



Afbeelding van <https://medium.com/koderlabs/introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63>

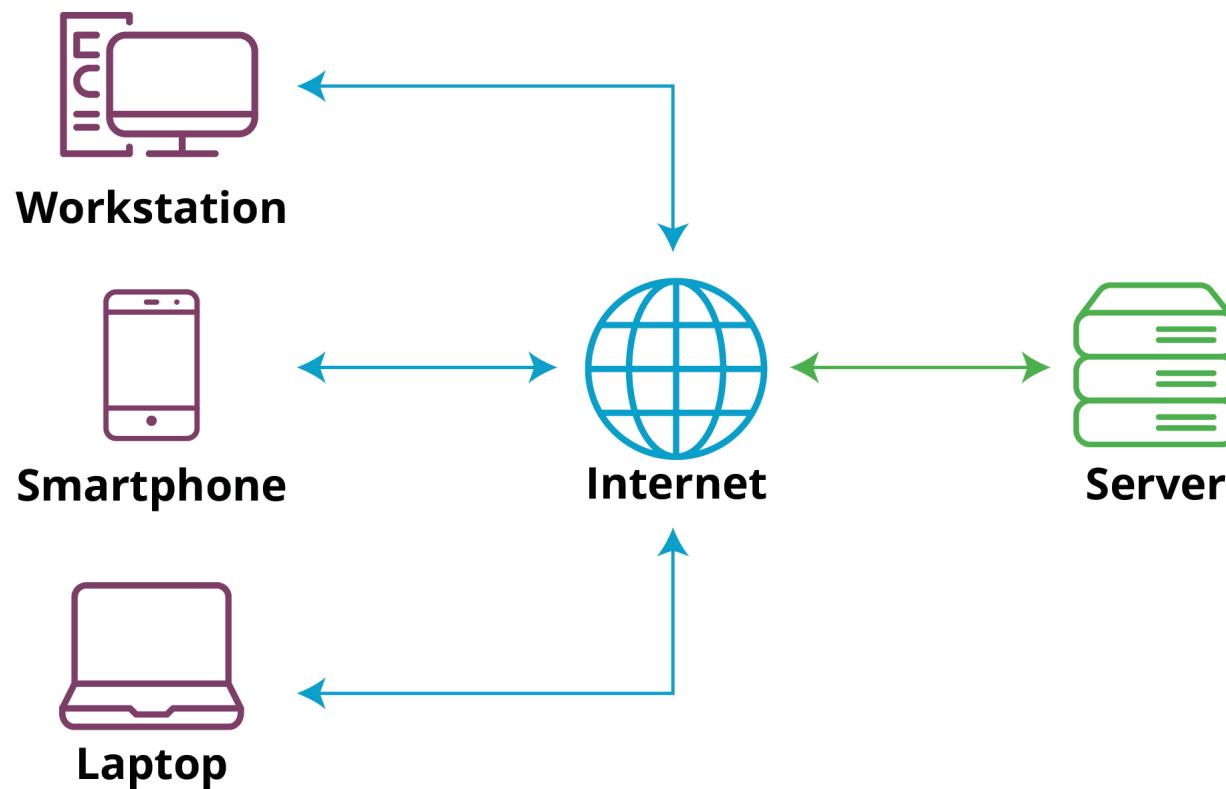
# Conclusie

- Monolithische architecturen zijn nog steeds een geldige keuze voor bepaalde soorten projecten, vooral waar eenvoud en cohesie belangrijk zijn en waar de applicatie niet verwacht wordt snel te groeien in complexiteit of schaal.
- Voor grotere, complexere systemen waar flexibiliteit, schaalbaarheid en onderhoudbaarheid kritiek zijn, kunnen alternatieve architecturale modellen zoals service-georiënteerde architecturen beter geschikt zijn.

# Client-servermodel

Architecturale modellen

# Client-servermodel



Afbeelding van <https://www.liquidweb.com/blog/client-server-architecture/>

# Client-servermodel

---

Het client-server architectuurmodel verdeelt de softwaretoepassing in twee hoofdcomponenten: de client en de server.

---

Clients sturen verzoeken naar de server, die deze verzoeken verwerkt en de vereiste gegevens of diensten terugstuurt.

---

Dit model maakt schaalbaarheid en gedistribueerde verwerking mogelijk, waardoor het geschikt is voor toepassingen met een groot aantal clients.

# Rolverdeling

## Client

- De client is typisch een computer of softwareapplicatie die toegang zoekt tot bepaalde diensten of middelen.
- Clients initiëren verzoeken naar servers voor informatie of diensten en wachten op hun reactie.

## Server

- De server is de computer of softwareapplicatie die diensten levert aan de clients.
- Het behandelt de verzoeken van meerdere clients en stuurt de gevraagde data of diensten terug.

# Communicatie

- De interactie tussen client en server vindt plaats via een netwerk waarbij het protocol afhankelijk is van de toepassing (bijvoorbeeld HTTP voor webtoepassingen, SMTP voor e-maildiensten).
- De client maakt een verzoek om een resource (zoals een webpagina of database query), de server verwerkt het verzoek en stuurt een reactie terug.

# Voorbeelden van het client-servermodel

## Webbrowser en webserver

- De webbrowser (client) vraagt een webpagina op van een webserver. De server vindt de pagina en stuurt deze terug naar de browser.

## E-mail clients en e-mail servers

- Een e-mail client zoals Outlook verzendt een verzoek naar de e-mail server om nieuwe berichten op te halen; de server controleert op nieuwe berichten en stuurt deze terug naar de client.

# Voordelen van het client-servermodel

- **Centraal beheer:**

- Servers kunnen centraal worden beheerd, wat betekent dat updates, back-ups, en beveiligingsmaatregelen allemaal vanuit één punt kunnen worden geregeld.

- **Schaalbaarheid:**

- Het is relatief eenvoudig om extra servers toe te voegen of de capaciteit van bestaande servers te vergroten om meer clients te kunnen bedienen.

- **Efficiëntie:**

- Servers kunnen heel krachtig zijn en in staat om veel clients tegelijk te bedienen, wat het gebruik van middelen efficiënter maakt.



# Nadelen van het client-servermodel

- **Afhankelijkheid van server:**

- Als de server om welke reden dan ook uitvalt, kunnen alle clients die afhankelijk zijn van die server niet meer functioneren.

- **Schaalbaarheidsbeperkingen:**

- Hoewel schaalbaar, kunnen grote aantallen gelijktijdige clientverzoeken leiden tot serveroverbelasting.

- **Beveiligingsrisico's:**

- Centrale servers kunnen een doelwit zijn voor cyberaanvallen, wat vereist dat strenge beveiligingsmaatregelen worden getroffen.



# Conclusie

- Het client-servermodel blijft een dominante structuur voor het bouwen van netwerkapplicaties dankzij zijn flexibiliteit, schaalbaarheid, en de efficiëntie in het beheren van middelen en diensten.
- Het is van cruciaal belang in de structurering van het internet en veel andere netwerktoepassingen die we dagelijks gebruiken.



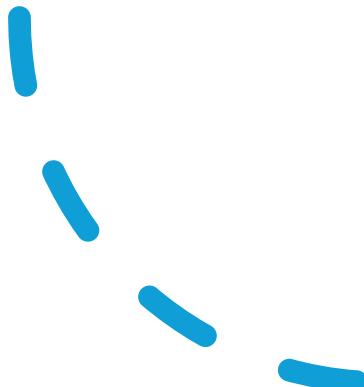
# Gelaagde architectuur

Architecturale modellen



# Gelaagde architectuur

- De gelaagde architectuur, ook bekend als n-tier architectuur, is een veelgebruikt ontwerppatroon in softwareontwikkeling dat gebruikt wordt om de structuur van een applicatie te organiseren door deze op te delen in afzonderlijke lagen, elk met hun eigen specifieke verantwoordelijkheid.



# Lagen

- Elke laag in de architectuur heeft een specifieke set van taken en verantwoordelijkheden.

## Presentation tier

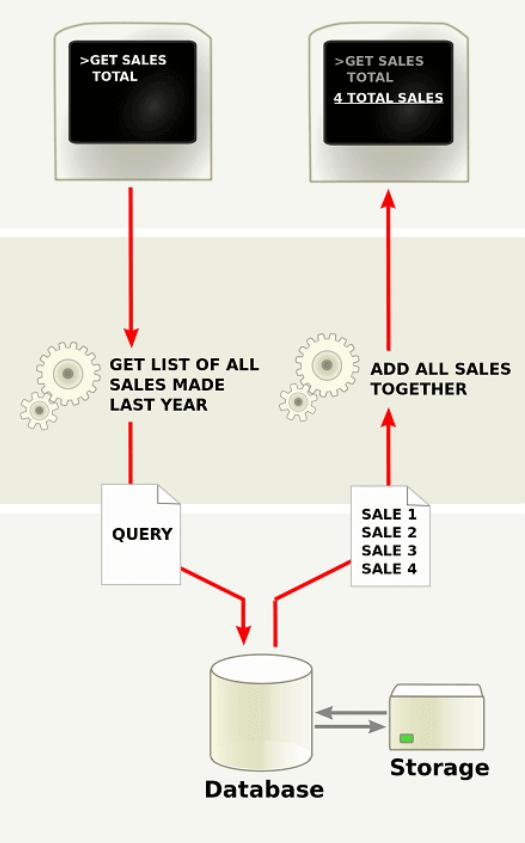
The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

## Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

## Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.





# Lagen

- Een typische gelaagde applicatie kan worden verdeeld in de volgende lagen:
  - **Presentatielaag (User interface):** Verantwoordelijk voor het weergeven van gebruikersinterface-elementen en het verwerken van gebruikersinteracties.
  - **Business Logica Laag (Service layer):** Deze laag verwerkt de gegevens, maakt beslissingen, en voert de kernfunctionaliteiten van de applicatie uit.
  - **Data Access Laag (DAL):** Verantwoordelijk voor communicatie met de database of andere opslagmechanismen om gegevens op te halen en op te slaan.
  - **Database Laag:** Dit is waar de data daadwerkelijk wordt opgeslagen. Deze laag is vaak een SQL-database of een andere vorm van dataopslag.

# Hoe werken de lagen samen?

- Elke laag communiceert alleen met de laag direct boven of onder het.
- Dit betekent dat de presentatielaag praat met de business logica laag, en de business logica laag praat met de data toegang laag, en zo verder.
- Deze manier van organiseren helpt om je code georganiseerd en duidelijk te houden. Als je iets wilt veranderen in hoe de gegevens worden weergegeven, hoef je alleen de presentatielaag aan te passen, zonder dat je zorgen hoeft te maken over de rest van de applicatie.



# Voorbeelden van gelaagde architectuur

- **Webapplicaties**

- Veel webapplicaties gebruiken een drie-lagen model bestaande uit een front-end presentatielaag (HTML, CSS, JavaScript), een back-end business logica laag (Java, Python, Ruby), en een database.

- **Enterprise Applications**

- Grottere enterprise systemen kunnen meerdere lagen bevatten, inclusief aparte lagen voor beveiliging, resource management, en meer.



# Voordelen van gelaagde architectuur

- **Modulariteit:**
  - Door het opsplitsen van applicaties in verschillende lagen, kunnen ontwikkelaars onafhankelijk aan verschillende delen van de applicatie werken zonder invloed op andere modules.
- **Onderhoudbaarheid:**
  - Het is makkelijker om een goed gestructureerde gelaagde applicatie te onderhouden en te updaten, omdat veranderingen in één laag minimale tot geen impact hebben op de andere lagen.
- **Herbruikbaarheid:**
  - Componenten binnen elke laag kunnen hergebruikt worden in verschillende delen van de applicatie of zelfs in verschillende projecten.
- **Flexibiliteit en schaalbaarheid:**
  - Het is relatief eenvoudig om de applicatie te schalen door lagen te repliceren of door de architectuur aan te passen zonder de gehele applicatie te hoeven herontwerpen.



# Nadelen van gelaagde architectuur

- **Prestatie issues:**

- Elke aanvraag moet mogelijk door meerdere lagen gaan voordat een operatie voltooid is, wat kan resulteren in een lagere performance in vergelijking met minder gescheiden architecturen.

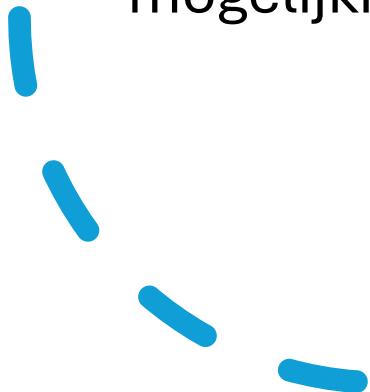
- **Complexiteit:**

- Hoewel het model helpt om de applicatie georganiseerd te houden, kan de initiële setup en configuratie complex zijn, vooral als je veel lagen hebt.



# Conclusie

- Gelaagde architectuur is een krachtig ontwerppatroon dat breed toegepast wordt in verschillende soorten softwareprojecten, van eenvoudige webapplicaties tot complexe enterprise systemen.
- Het biedt een duidelijke structuur voor ontwikkelaars om te volgen en bevordert schaalbaarheid, onderhoudbaarheid en de mogelijkheid tot onafhankelijke ontwikkeling



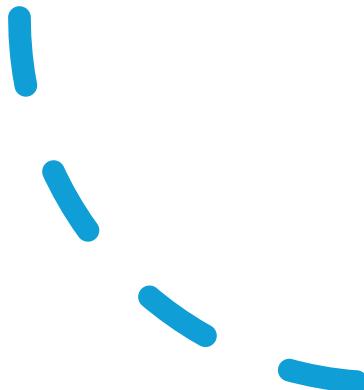
# Service-georiënteerde architectuur

Architecturale modellen



# Service-georiënteerde architectuur

- Service-georiënteerde architectuur is een manier om software te bouwen waarbij je verschillende functies van de software opdeelt in aparte eenheden, genaamd 'services'.
- Deze services zijn als kleine, onafhankelijke mini-programma's die elk een specifieke taak uitvoeren.





# Wat zijn services?

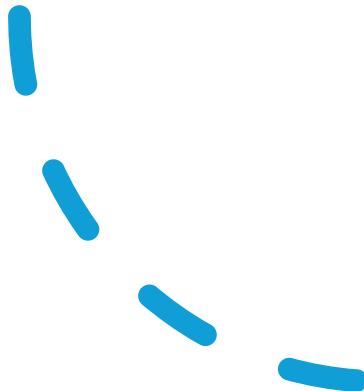
- In de context van SOA, kun je een service zien als een klein programma of een stukje code dat een bepaalde taak heel goed doet.
- Bijvoorbeeld, je kunt een service hebben die de temperatuur meet, een andere die het weerbericht controleert, en nog een die nieuwsberichten laat zien.
- Deze services werken onafhankelijk van elkaar, maar kunnen met elkaar communiceren.





# Hoe werken services samen?

- SOA gebruikt netwerkprotocollen om deze services te laten communiceren.
- Dit betekent dat een service een verzoek kan sturen naar een andere service en een antwoord terug kan krijgen. Deze communicatie gebeurt vaak via het internet.



# Waarom is SOA handig?

- **Flexibiliteit:**
  - Omdat elke service onafhankelijk is, kun je gemakkelijk veranderingen maken in één service zonder dat je de hele software moet aanpassen. Dit maakt het bijwerken van je software veel makkelijker.
- **Herbruikbaarheid:**
  - Services kunnen in verschillende programma's of door verschillende gebruikers hergebruikt worden. Dit betekent dat als je eenmaal een goede service hebt ontwikkeld, je deze op veel plaatsen kunt gebruiken, wat tijd en moeite bespaart.
- **Schaalbaarheid:**
  - Je kunt meer exemplaren van dezelfde service toevoegen om aan hogere eisen te voldoen zonder dat dit problemen oplevert voor andere delen van je software. Dit is erg handig wanneer veel mensen tegelijkertijd je applicatie willen gebruiken.

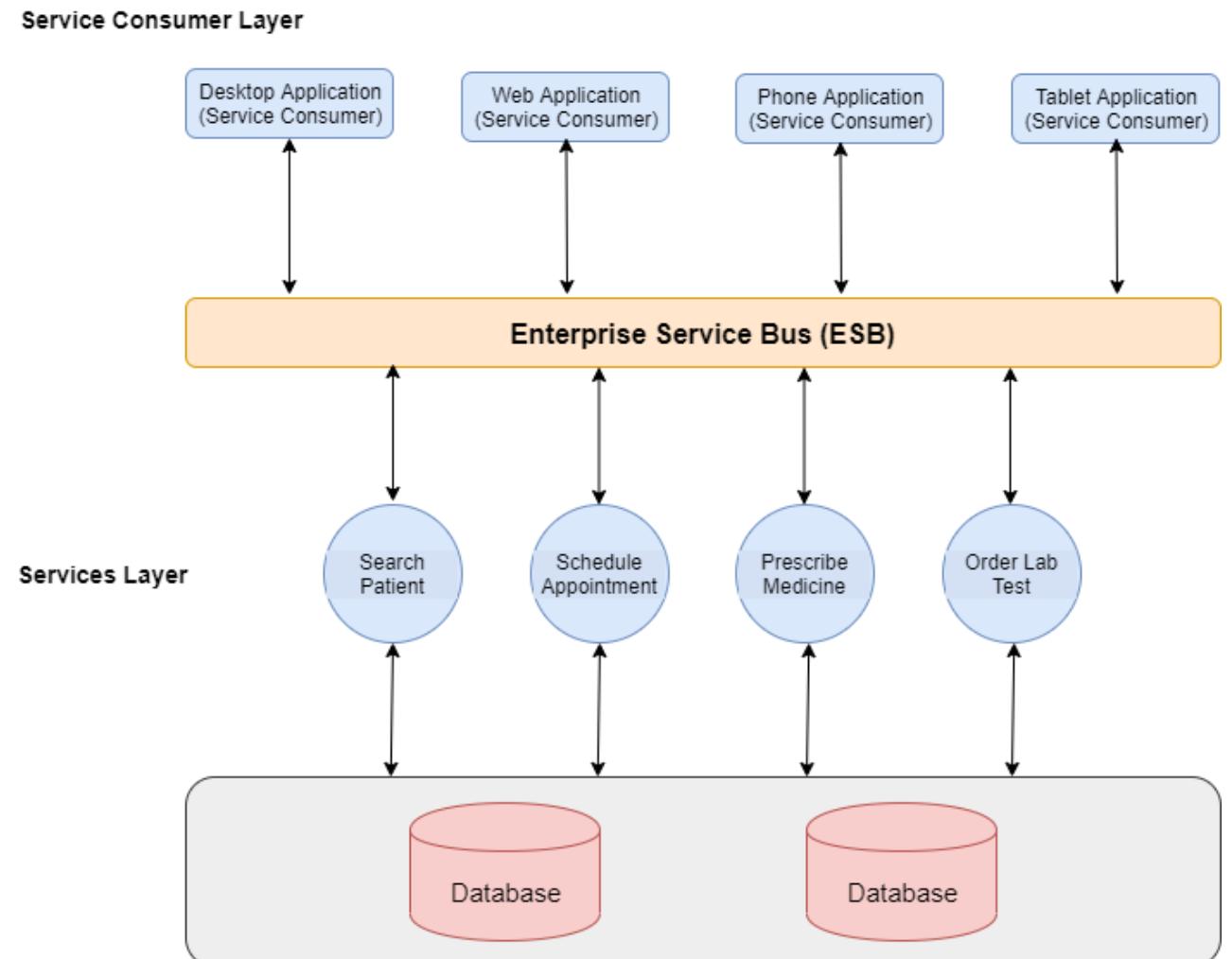


# Voorbeeld

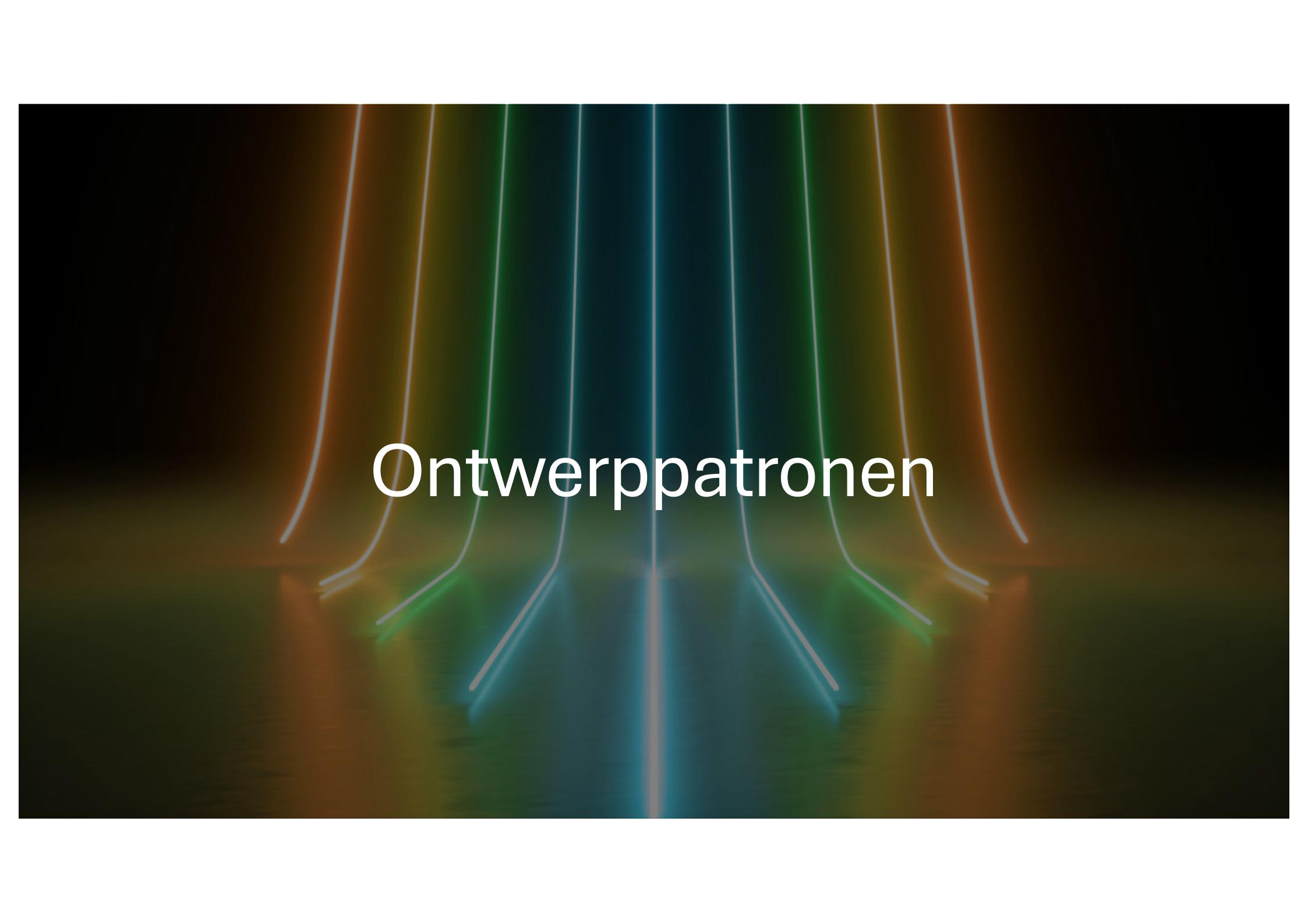
- Stel je een online winkel voor. De website kan verschillende services gebruiken: één om zoekopdrachten van klanten te verwerken, één om voorraad te controleren, één om betalingen te verwerken, enzovoort.
- Elk van deze services werkt onafhankelijk maar ze werken samen om je een soepele winkelervaring te bieden.



# Voorbeeld



Afbeelding van <https://medium.com/@adeelsarwarblog/what-is-service-oriented-architecture-9b4406fcdeb>



# Ontwerppatronen

# Wat zijn ontwerppatronen?

---

Ontwerppatronen (design patterns) zijn een cruciaal concept in software engineering, ontworpen om typische problemen op te lossen die ontwikkelaars tegenkomen bij het ontwerpen van softwareapplicaties.

---

Deze patronen bieden bewezen oplossingen in de vorm van templates die gebruikt kunnen worden om algemene ontwerproblemen op een efficiënte en herbruikbare manier aan te pakken.

# Wat zijn ontwerppatronen?

---

Ontwerppatronen zijn in essentie een set van standaard richtlijnen, bewezen door de ervaring van verschillende softwareontwikkelaars, om bepaalde problemen in softwareontwerp op te lossen.

---

Deze patronen zijn niet kant-en-klare codes die je in je programma kunt plakken, maar concepten die je kunt gebruiken als blauwdruk om je eigen oplossingen te ontwerpen.

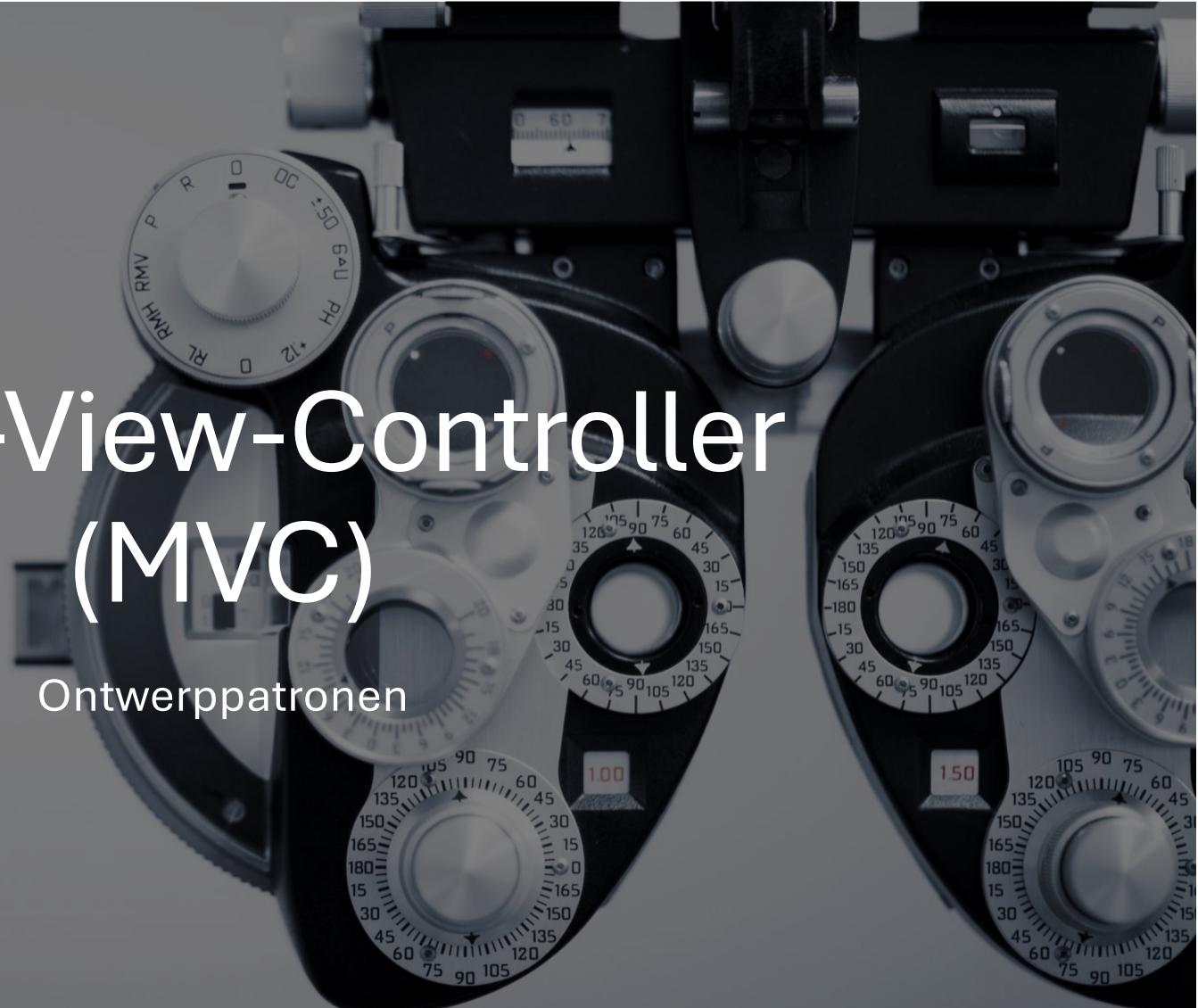


# Waarom zijn ontwerppatronen belangrijk?

- **Herbruikbaarheid:**
  - Ze bieden oplossingen die in veel verschillende situaties kunnen worden toegepast, waardoor ontwikkelaars tijd besparen die ze anders aan het oplossen van dezelfde problemen zouden besteden.
- **Best practices:**
  - Ontwerppatronen zijn gebaseerd op de collectieve ervaring van ontwikkelaars en vertegenwoordigen best practices in softwareontwerp.
- **Communicatie:**
  - Ze bieden een gemeenschappelijke taal voor ontwikkelaars. Als je zegt dat je een "Singleton" of "Factory" patroon gebruikt, hebben andere ontwikkelaars met kennis van patronen meteen een goed idee van wat je bedoelt.

# Model-View-Controller (MVC)

Ontwerppatronen



# Model-View-Controller (MVC)

---

MVC, wat staat voor Model-View-Controller, is een ontwerppatroon dat vaak gebruikt wordt in de ontwikkeling van user interfaces.

---

Het helpt bij het organiseren van code op een manier die taken scheidt en het makkelijker maakt om je code te beheren en uit te breiden.

---

Het is een van de meest bekende en veelgebruikte ontwerppatronen in de ontwikkeling van software, vooral voor webapplicaties en desktop grafische gebruikersinterfaces (GUIs).

---

Dit patroon is bijzonder populair in webapplicaties en wordt ondersteund door veel ontwikkelingsframeworks.

# MVC splitst een applicatie op in drie onderdelen

- **Model:**

- Dit is het deel van je applicatie dat de kernfunctionaliteiten en data beheert. Het model is verantwoordelijk voor het ophalen van data uit de database, het verwerken van die data, en het beheren van de gegevensregels, business logic, en berekeningen. Belangrijk is dat het model onafhankelijk is van de gebruikersinterface.

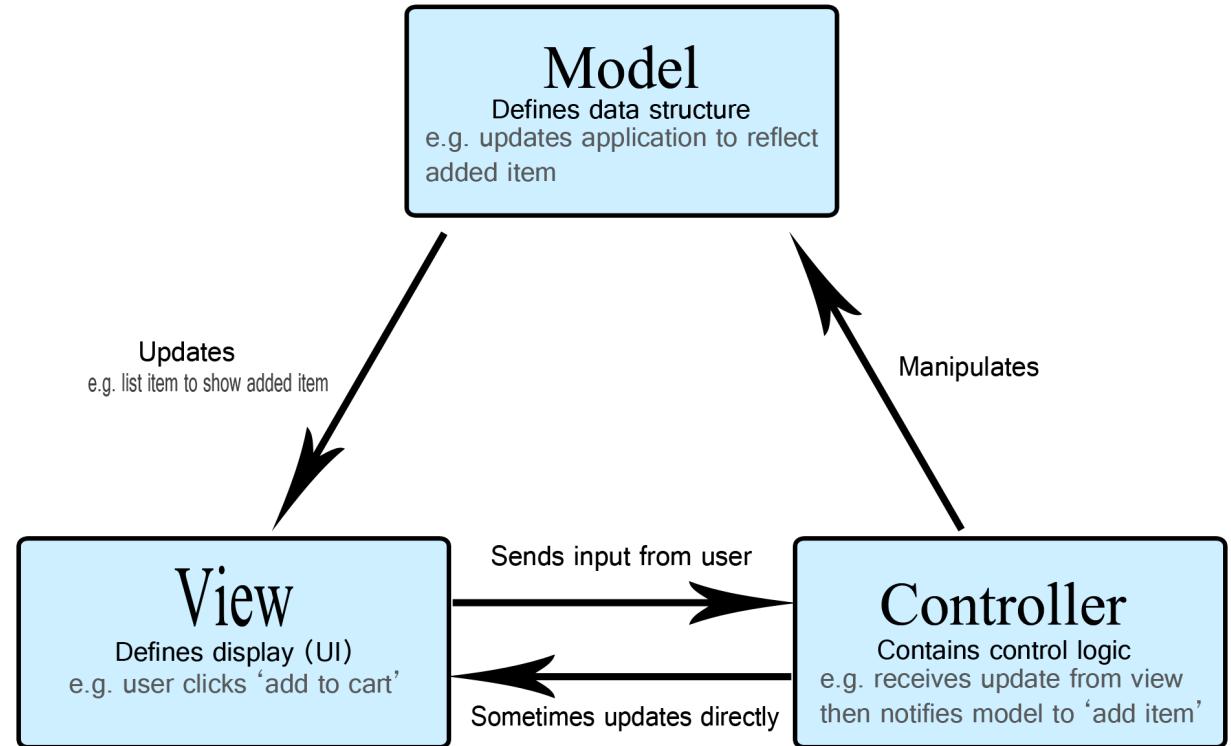
- **View:**

- Dit is het deel dat verantwoordelijk is voor het tonen van de output aan de gebruiker. In een webapplicatie omvat dit de HTML, CSS, en JavaScript die gebruikt worden om de inhoud te tonen.

- **Controller:**

- De Controller fungeert als een tussenpersoon tussen het Model en de View. Het is de bemiddelaar die de input van de gebruiker opneemt, verwerkt en dan beslist welke data uit het model nodig is en welke view er gebruikt moet worden om die data te tonen.

# Model-View-Controller (MVC)

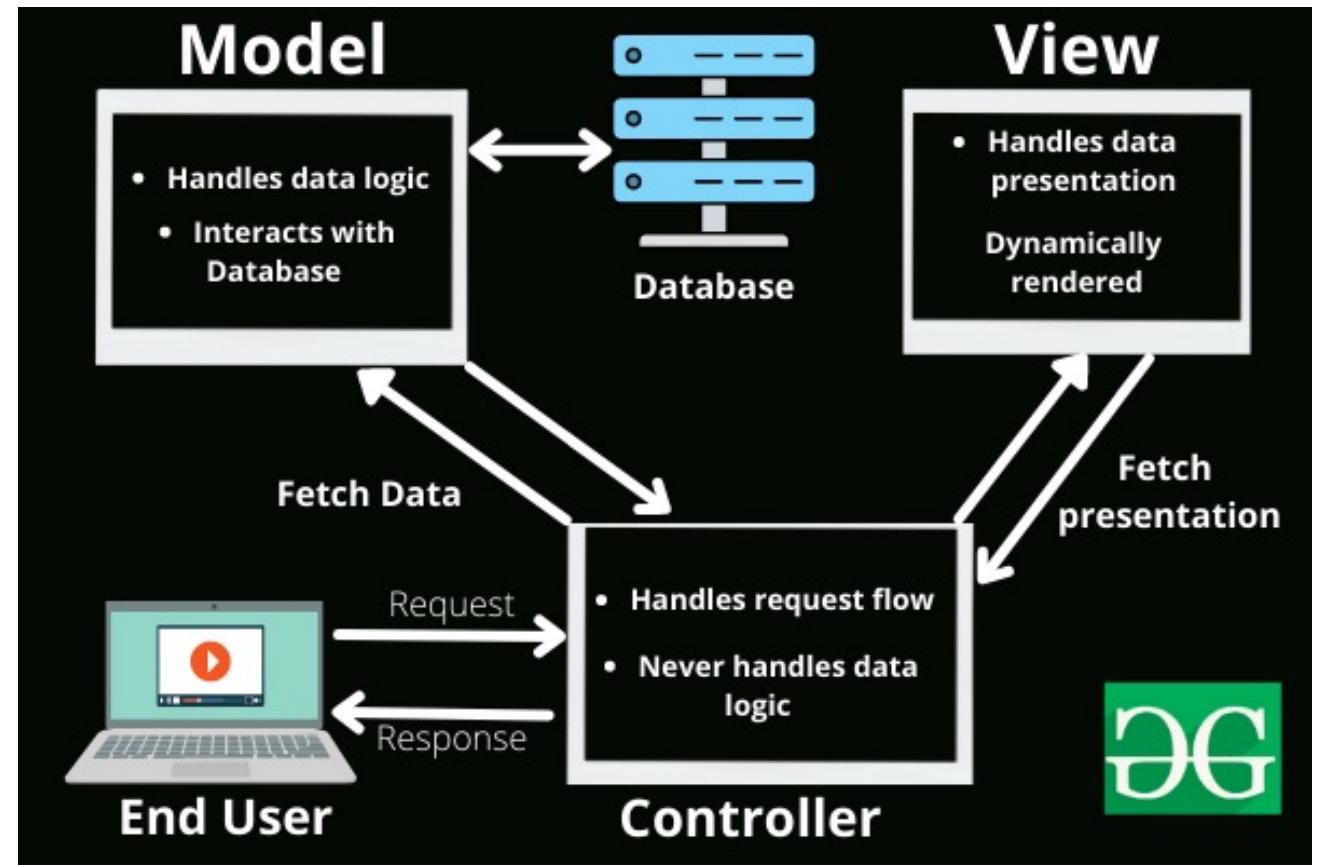




# Hoe werkt MVC?

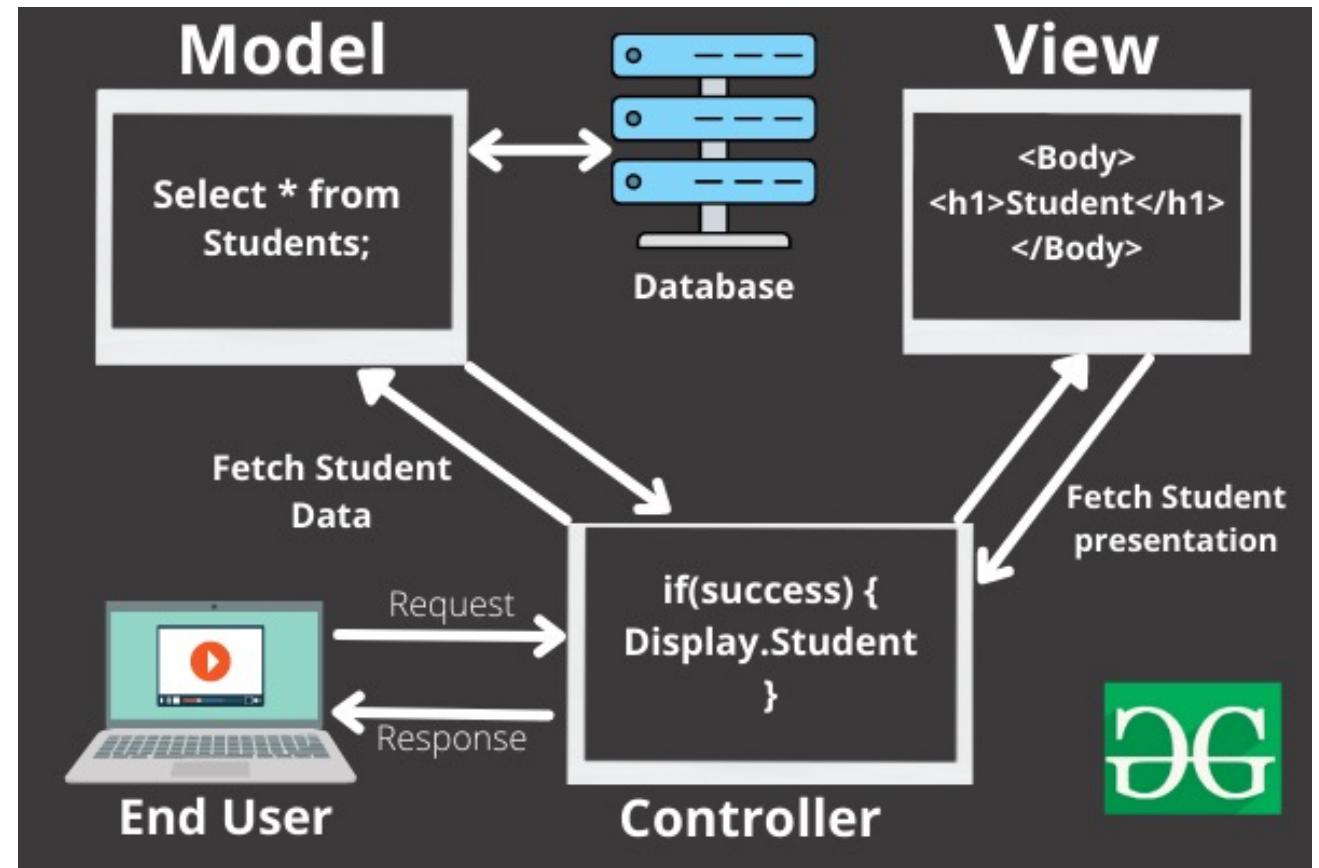
1. De gebruiker doet een verzoek aan de applicatie (bijvoorbeeld door op een link te klikken of een adres in te voeren).
2. De Controller ontvangt dit verzoek en beslist welke acties er genomen moeten worden.
3. De Controller kan bijvoorbeeld data ophalen uit het Model.
4. De Controller kiest een View en geeft de data aan deze View om te worden getoond.
5. De View toont de data aan de gebruiker.

# Model-View-Controller (MVC)



<https://www.geeksforgeeks.org/mvc-framework-introduction/>

# Model-View-Controller (MVC)



<https://www.geeksforgeeks.org/mvc-framework-introduction/>

# Voordelen van MVC

- **Scheiding van verantwoordelijkheden:**
  - Duidelijk gedefinieerde rollen voor Model, View, en Controller zorgen voor een georganiseerde codebasis die makkelijker te onderhouden, te testen, en te ontwikkelen is.
- **Flexibiliteit:**
  - Het is makkelijk om de gebruikersinterface te veranderen zonder dat je de logica van de dataverwerking hoeft te wijzigen, dankzij de scheiding tussen View en Model.

# Enkele begrippen

# Wat is frontend development

- Frontend verwijst naar het deel van een website of applicatie dat de gebruikers direct zien en mee interactie hebben. Het omvat alles wat met de gebruikersinterface en gebruikerservaring te maken heeft.
- De taak van een frontend ontwikkelaar is om ontwerpen om te zetten in realiteit op het scherm en ervoor te zorgen dat de applicatie er goed uitziet en goed werkt op verschillende apparaten en browsers.

# Technologieën in frontend development

- HTML (HyperText Markup Language): De basis van elke webpagina, gebruikt voor het structureren van content.
- CSS (Cascading Style Sheets): Gebruikt voor het stylen van de webpagina, inclusief lay-outs, kleuren, en lettertypes.
- JavaScript: Maakt webpagina's interactief en dynamisch. Frameworks zoals React, Angular, en Vue.js zijn populair voor geavanceerde taken.

# Wat is backend development?

---

Backend verwijst naar de serverzijde van een website of applicatie. Het is het deel dat gebruikers niet zien, maar dat essentieel is voor de functionaliteit van de site.

---

Backend ontwikkeling gaat over het bouwen en onderhouden van de technologie die nodig is om de data-verwerkende operaties van een applicatie te beheren. Het omvat het beheren van databases, serverlogica, en de integratie van verschillende applicatieprocessen.

# Technologieën in backend development

- Server Talen: Python, Ruby, Java, PHP, .NET, Node.js, etc.
- Databases: MySQL, MongoDB, PostgreSQL, Oracle, etc.
- Server Management: Kennis van hoe servers werken, hoe om te gaan met API's (Application Programming Interfaces), en hoe netwerkverzoeken worden afgehandeld.

# Wat is full stack development?

- Full Stack Development combineert zowel frontend als backend development.
- Een full stack ontwikkelaar heeft de vaardigheden om zowel client-side als server-side onderdelen van een webapplicatie te bouwen en te onderhouden.
- Dit betekent dat ze kunnen werken aan het ontwerpen van de gebruikersinterface, het coderen van de serverlogica, en het configureren van de database.

# Voordelen van een full stack ontwikkelaar

- **Veelzijdigheid:** Ze kunnen werken aan verschillende aspecten van een project, van de user interface tot de database.
- **Probleemoplossing:** Full stack ontwikkelaars kunnen problemen overzien en oplossen door het hele project, omdat ze een brede kennisbasis hebben.
- **Projectmanagement:** Hun vermogen om zowel frontend als backend taken te beheren maakt het makkelijker om projecten te leiden en te coördineren.