Background pattern

Description automatically generated

# intro

Vue is een single-page application framework. Dit betekent dat slechts één enkele pagina in de browser wordt ingeladen en dat de inhoud van deze pagina dynamisch zal veranderen op basis van de input van de bezoeker op de site.

Een groot voordeel van single-page applications is een betere gebruikerservaring. De gebruiker zal geen pagina-refreshes zien, bijvoorbeeld als er naar een andere pagina gesurft wordt, waardoor het surfen door de pagina een stuk vloeiender aanvoelt.

Vue biedt echter nog veel meer voordelen dan alleen een betere gebruikerservaring. Het maakt het ontwikkelen van een site gemakkelijker door vele zaken die anders nogal ingewikkeld kunnen zijn, zoals bijvoorbeeld routing of het beheren van de state, eenvoudiger te maken door een deel voorgeprogrammeerde tools aan te bieden die deze taken voor je uitvoeren. Hierdoor hoef je deze zaken niet meer volledig zelf te schrijven.

Verder maakt Vue gebruik van zijn eigen templating language, waarmee het heel eenvoudig wordt om dynamische data en content op bepaalde plaatsen in je pagina in te voegen.

Bovendien maakt Vue het heel eenvoudig om op een simpele manier data tussen componenten door te geven. Het gebruik van componenten zorgt er op zijn beurt weer voor dat de code die je schrijft een stuk herbruikbaarder wordt. Zo kun je een component op meerdere plaatsen in je pagina hergebruiken en afhankelijk van de gegevens die je eraan meegeeft kan de inhoud binnen het component dan wijzigen.

# vue toevoegen aan een project

Vue kan op verschillende manieren aan je project worden toegevoegd.

* Als CDN package op de pagina
* Door de javascript files te downloaden en ze zelf te hosten
* Door vue te installeren via een package manager zoals npm
* Door [de officiële CLI](https://github.com/vuejs/create-vue) (command-line interface) te gebruiken om een standaard project op te starten.

Het is deze laatste manier die we in deze cursus gaan gebruiken om een project op te starten. Dit is de gemakkelijkste manier om aan een vue project te beginnen aangezien een hele berg configuratiewerk al reeds voor ons gedaan gaat worden. Moest je meer info willen over de andere manieren om vue aan een reeds bestaand project toe te voegen kun je [hier](https://v3.vuejs.org/guide/installation.html#release-notes) zeker eens een kijkje gaan nemen.

## Een nieuw vue project creëren

Om een nieuw vue project te creëren run je het volgende commando in een command prompt op de plaats waar je een vue project wilt aanmaken:



Duid vervolgens de volgende opties aan:

A computer screen shot of a program

Description automatically generated

Verklaring begrippen:

Typescript:   
TypeScript is een open-source programmeertaal en een superset van JavaScript. Het is ontwikkeld door Microsoft en is bedoeld om het makkelijker te maken om grootschalige JavaScript applicaties te bouwen, door functies toe te voegen zoals static typing, classes, interfaces en andere object-georiënteerde programmeerconcepten.

Met TypeScript kun je JavaScript code schrijven met optionele statische types, wat betekent dat je de datatypes van variabelen kunt specificeren op het moment dat je de code schrijft. Dit kan het makkelijker maken om fouten te vinden en je code robuuster en beter onderhoudbaar te maken. Bovendien biedt TypeScript een krachtige tooling voor ontwikkelaars, waardoor het makkelijker is om grote codebases te onderhouden en samen te werken in teams.

JSX:  
JSX is een syntaxisuitbreiding voor JavaScript die het mogelijk maakt om HTML-achtige code te schrijven binnen je JavaScript-code. Het wordt vaak gebruikt in React-toepassingen, maar kan ook gebruikt worden in combinatie met Vue.js.

Door het inschakelen van JSX-ondersteuning in je Vue.js-installatie, wordt jouw Vue-project geconfigureerd om het gebruik van JSX in je componenten te ondersteunen. Dit kan handig zijn als je de syntaxis van JSX verkiest of als je bestaande componenten die geschreven zijn in JSX opnieuw wilt gebruiken.

Vue router:  
De Vue Router is een officiële routing library voor Vue.js. Het biedt een mogelijkheid om eenvoudig client-side routing in je Vue-applicatie te implementeren, wat betekent dat het de URL van je applicatie dynamisch kan bijwerken zonder dat er een nieuwe pagina hoeft te worden geladen.

Het doel van de Vue Router is om het ontwikkelen van single-page applicaties (SPA's) in Vue.js te vergemakkelijken. Met de router kun je verschillende routes definiëren en deze koppelen aan specifieke componenten in je applicatie. Op die manier kun je een navigatiestructuur voor je applicatie creëren en specifieke componenten tonen wanneer de gebruiker een bepaalde URL bezoekt.

De Vue Router biedt ook andere handige functies, zoals nested routes, named routes, programmatic navigation en route guards. Dit maakt het gemakkelijk om complexe applicaties met meerdere views te bouwen en de navigatie en beveiliging van de applicatie te beheren.

Pinia:  
Pinia is een state management library die speciaal is ontworpen voor gebruik met Vue 3. Het biedt een eenvoudige, maar krachtige manier om de state van je Vue-applicatie te beheren. In plaats van te werken met grote, gecentraliseerde stores, maakt Pinia gebruik van kleine, onafhankelijke stores die elk een specifiek deel van de state van je applicatie beheren. Deze stores zijn gemakkelijk te maken en te gebruiken, en kunnen gemakkelijk worden gecombineerd om complexe state-management situaties op te lossen.

Pinia maakt ook gebruik van de reactiviteitssysteem van Vue, waardoor je state changes eenvoudig kunt bijhouden en automatisch kunt reageren op veranderingen in je applicatie. Hierdoor is Pinia een goede keuze voor het beheren van de state van grote en complexe Vue-applicaties, waarbij je de controle wilt houden over de state van verschillende delen van je applicatie.

Unit testing en end-to-end testing:  
Unit testing en end-to-end testing zijn twee verschillende benaderingen om de kwaliteit van software te verbeteren:

Unit testing is het proces van het testen van individuele delen (eenheden) van de code om ervoor te zorgen dat ze correct werken en voldoen aan de specificaties. Het doel van unit testing is om te zorgen dat de afzonderlijke onderdelen van de code correct werken, zodat de kans op fouten en bugs in de code wordt verkleind. Dit kan leiden tot een betere kwaliteit van de software en maakt het gemakkelijker om fouten op te sporen en op te lossen. Unit tests worden meestal automatisch uitgevoerd als onderdeel van een Continuous Integration (CI) / Continuous Deployment (CD) pipeline.

End-to-end testing is een methode om de functionaliteit van een applicatie te testen van begin tot eind. Dit houdt in dat het testen van de complete applicatie, inclusief de interfaces met de gebruiker, de database en eventuele andere systemen waarmee de applicatie communiceert. Het doel van end-to-end testing is om de applicatie te testen zoals een gebruiker het zou doen, om te controleren of alle onderdelen van de applicatie correct samenwerken en om te zorgen dat de applicatie goed werkt onder verschillende omstandigheden en gebruiksscenario's.

ESLint:  
ESLint is een open-source JavaScript linter die gebruikt wordt om programmeerfouten en stijlproblemen in JavaScript code te identificeren. Het is ontworpen om ontwikkelaars te helpen bij het schrijven van foutvrije en gestandaardiseerde code, ongeacht de omvang van het project.

Met ESLint kun je specifieke regels en configuraties instellen voor jouw JavaScript-project, en het zal dan je code scannen en een rapport genereren over eventuele problemen die zijn gevonden. Dit kan variëren van syntaxisfouten en variabelen die niet zijn gedefinieerd, tot stilistische problemen zoals inconsistent gebruik van inspringing of het niet volgen van bepaalde code conventies.

ESLint kan worden geïntegreerd in verschillende ontwikkelingsomgevingen, zoals bijvoorbeeld Visual Studio Code of WebStorm, en kan ook worden gebruikt als een pre-commit hook in Git om ervoor te zorgen dat je code consistent en correct is voordat deze wordt geüpload naar de repository. Het is een zeer nuttige tool voor ontwikkelaars om de kwaliteit en leesbaarheid van hun JavaScript-code te verbeteren.

Verdere stappen:

Text

Description automatically generated

Hier zie je hoe je jouw Vue-project kunt runnen.

Wanneer je het commando `npm install` uitvoert, zal NPM (Node Package Manager) de dependencies installeren die zijn gespecificeerd in het `package.json`-bestand in jouw Vue-project. Dit betekent dat NPM de benodigde packages downloadt en installeert, zodat je project goed kan werken.

Wanneer je daarna het commando `npm run dev` uitvoert, start NPM de `dev` script die in de `package.json`-file is gedefinieerd. Dit script is bedoeld om je Vue-project lokaal te draaien en je de mogelijkheid te geven om het te ontwikkelen. Dit script start meestal een lokale server op een bepaalde poort en opent automatisch een browser met jouw Vue-applicatie. Zo kun je de applicatie live testen en eventuele wijzigingen die je aanbrengt onmiddellijk zien.

Text

Description automatically generated

De development server kan gestopt worden door ctrl + c in te drukken.

Om te beginnen gaan we ons vue project openen in WebStorm. Open WebStorm en klik op File > Open en selecteer de directory die je net hebt aangemaakt.

# Een vue instantie aanmaken en mounten

In een Vue project zijn er twee belangrijke bestanden: index.html en main.js (main.ts als je voor typescript support hebt gekozen bij de installatie). De index.html is het startpunt van je applicatie en dit bestand bevat de structuur van de pagina. Hierin kun je bijvoorbeeld de basis HTML code plaatsen.

Het main.js bestand is het hart van je Vue applicatie. In dit bestand kun je bijvoorbeeld de Vue instance initialiseren, data en methodes definiëren en componenten registreren.

Om deze twee bestanden met elkaar te verbinden, moet je een script tag in de index.html plaatsen die verwijst naar het main.js bestand. Hiermee wordt de JavaScript code uit het main.js bestand uitgevoerd en wordt de functionaliteit aan de pagina toegevoegd.

### Main.js

Text

Description automatically generatedBinnen de main.js file gebeuren een aantal belangrijke zaken:

* De **createApp** functie wordt geïmporteerd vanuit de **vue** module.
* De **createPinia** functie wordt geïmporteerd vanuit de **pinia** module.
* De **App** component wordt geïmporteerd vanuit het **App.vue** bestand.
* De **router** instance wordt geïmporteerd vanuit het **router.js** bestand.
* De **main.css** stylesheet wordt geïmporteerd.

De **import** statement in JavaScript wordt gebruikt om code vanuit andere bestanden of modules te importeren en beschikbaar te maken in het huidige bestand.

Er wordt vervolgens een nieuwe Vue applicatie instance aangemaakt met de **createApp** functie. Door de **App** component als parameter mee te geven wordt dit de meest bovenliggende component die gerenderd wordt in onze vue applicatie. Daarna worden de **Pinia** state management library en de router instance geregistreerd bij de Vue applicatie instance.

De Vue applicatie instance wordt uiteindelijk gemount binnen het element met de ID **app** in het index.html bestand.

Text

Description automatically generated with medium confidence

# De vue component

Binnen Vue gaan we werken met herbruikbare componenten. Het herhalen van grote brokken HTML die steeds terugkomen op al je verschillende pagina's is hierdoor dus verleden tijd. Vue-componenten bestaan uit drie delen:

### Template

De **<template>**-tag in Vue.js bevat de HTML-markup en Vue-specifieke templating syntax die samen de visuele representatie van een component vormen. Dit betekent dat je binnen de `<template>`-tag alle elementen en componenten kunt gebruiken die je nodig hebt om de UI van jouw component vorm te geven.

Je kunt ook Vue's template syntax gebruiken om data te binden aan je HTML, zoals het weergeven van variabelen of het uitvoeren van condities en loops. Vue.js gebruikt een Virtual DOM om veranderingen in de template snel en efficiënt bij te werken, waardoor de UI van je applicatie altijd up-to-date is.

### Script

De **<script>**-tag in Vue.js bevat de logica van een component, inclusief de data, methods, lifecycle hooks en event-handlers. Hierin kun je de eigenschappen en methoden definiëren die de basis vormen van jouw component, en deze koppelen aan de bijbehorende HTML-template.

Er zijn twee manieren om deze logica te definiëren in Vue.js: de Options API en de Composition API. In de Options API definieer je alles in één groot object, terwijl in de Composition API je de logica opsplits in kleinere, herbruikbare stukjes code.

Kortom, de `<template>`-tag in Vue.js bevat de visuele representatie van een component, terwijl de `<script>`-tag de logica bevat die de component gedraagt en functioneert. Samen vormen deze twee tags de basis van elke Vue-component en stellen ze je in staat om krachtige en dynamische UI's te bouwen voor jouw webapplicatie.

### Style

De **<style>**-tag in Vue.js bevat alle CSS-stijlen die van toepassing zijn op de HTML in de **<template>**-tag van een component. Dit kunnen standaard CSS-regels zijn, maar ook CSS-preprocessors zoals Sass of Less.

Het voordeel van het definiëren van CSS-stijlen in de **<style>**-tag van een component in plaats van in een externe stylesheet, is dat je alle stijlen voor een bepaald component op één centrale plek kunt definiëren en beheren. Dit maakt het gemakkelijker om de stijlen te begrijpen en te onderhouden.

In Vue.js kun je de stijlen in de **<style>**-tag ook koppelen aan specifieke onderdelen van de HTML in de **<template>**-tag, door gebruik te maken van Vue's CSS-scoping-mechanisme. Hierdoor kunnen stijlen van het ene component niet per ongeluk van invloed zijn op andere componenten in de applicatie, en vice versa.

Kortom, de **<style>**-tag in Vue.js bevat alle CSS-stijlen die van toepassing zijn op de HTML van een component, en maakt het gemakkelijk om deze stijlen centraal te definiëren en te beheren.

## Options & composition api

In Vue kun je componenten maken met behulp van de Options API of de Composition API. Dit zijn twee verschillende manieren om componenten te schrijven, elk met hun eigen voor- en nadelen:

Options API: De Options API is de standaard manier om componenten te schrijven in Vue. Met deze API worden componenten geschreven als een object met verschillende opties, zoals data, computed, methods, watch, enzovoort. De Options API is eenvoudig te begrijpen en is geschikt voor kleine tot middelgrote projecten.

Composition API: De Composition API is een nieuwe manier om componenten te schrijven in Vue 3. Met deze API worden componenten geschreven als functies die gebruikmaken van andere functies om logica te delen en te hergebruiken. De Composition API maakt het gemakkelijker om complexe logica te schrijven en te hergebruiken, vooral in grote projecten. De Composition API heeft ook voordelen op het gebied van code organisatie, type checking en code splitting.

Kortom, de Options API is eenvoudig te begrijpen en geschikt voor kleine tot middelgrote projecten. De Composition API is meer geschikt voor grotere projecten met complexe logica, maar is ook iets complexer om te begrijpen. Binnen deze cursus zullen we de options API gebruiken.

Als je de [vue documentatie](https://vuejs.org/guide/introduction.html) online bekijkt, let dan zeker op dat je de documentatie van de juiste API aan het bekijken bent. Deze toggle staat linksboven aan de pagina.

Graphical user interface, application

Description automatically generated

### App.vue

Vooraleer we verder kunnen moeten we eerst de App.vue file nog veranderen van de composition API naar de options API aangezien de door de create-vue library aangemaakte bestanden in de composition API gemaakt zijn. Deze code kunnen jullie in [de bijlage](#_App.vue_options_api) terugvinden. Kopieer de code en vervang die met de code in jouw App.vue bestand.

Text

Description automatically generatedIn het vorige hoofdstuk zagen we hoe we binnen main.js een nieuwe vue app creëren via de createApp methode en hierbinnen het App.vue component gaan renderen. Laten we nu wat dieper ingaan op die App.vue component.

Binnen de template tags komt de vue template language te staan. Hier staat de HTML die gerenderd zal worden zodra App.vue gerenderd wordt.

App.vue zal een image, een HelloWorld component, 2 router-links en een router-view component renderen. Op deze laatste 2 gaan we zometeen verder in gaan.   
Let wel op, de HelloWorld component is ook nog met de composition API geschreven. Dit is geen probleem, de 2 API’s kunnen door elkaar gebruikt worden, maar we gaan deze niet veranderen naar options API omdat we daar toch niks aan gaan veranderen in deze cursus. Weldra hebben we deze component nooit meer nodig.

In de script tag komt de logica van de component. Voorlopig bevat die enkel een components property waar we alle componenten die we in onze template willen renderen moeten plaatsen.

# vue Router

De Vue Router is een officiële plugin voor Vue.js die je kunt gebruiken om client-side routing aan je Vue.js-applicatie toe te voegen. Dit betekent dat je verschillende "pagina's" in je applicatie kunt definiëren, en deze via de URL kunt openen zonder de pagina te hoeven refreshen.

In de Vue Router kun je routes definiëren, die elk overeenkomen met een bepaalde URL. Elke route bevat een component dat moet worden gerenderd wanneer de route wordt geopend. De Vue Router zorgt ervoor dat het juiste component wordt gerenderd binnen de router-view op basis van de huidige URL.

Text

Description automatically generatedOnder de src map (directory) staat de map router, hierbinnen kun je de file index.js terugvinden. Hier worden de verschillende routes van de vue-router gedefinieerd.

Op regel 4 wordt een instantie van de vue-router gecreëerd via de createRouter methode die we importeren uit de vue-router package op regel 1. Aan deze methode geven we een object met de opties die we in willen stellen in onze router mee. Op dit moment bevat dit object de [history-mode](https://router.vuejs.org/guide/essentials/history-mode.html#html5-history-mode) die we willen gebruiken binnen de router en een array met de verschillende routes die we willen voorzien in de app.

Zoals je kan zien is elke nieuwe route een afzonderlijk object binnen de routes array. Elke route object moeten we voorzien van een path, een name en vervolgens geven we mee welk component er gerenderd moet worden in het ***router-view*** component (App.vue regel 15) indien er naar die route genavigeerd wordt.

In het eerste route object zien we dat de ***Home*** component gerenderd moet worden als naar de ´/´-route genavigeerd wordt. Welk component dat ***Home*** component is kun je zien op regel 2, waar de Home component geïmporteerd wordt.

### Router-link

Text

Description automatically generated

De **router-link**-component is een onderdeel van de Vue Router, en stelt je in staat om links te maken binnen je Vue.js-applicatie die overeenkomen met de gedefinieerde routes. Je kunt een **router-link**-component bijvoorbeeld gebruiken om van de ene pagina naar de andere te navigeren, zonder dat de hele pagina hoeft te worden vernieuwd.

Het **to**-attribuut van de **router-link**-component bepaalt welke route moet worden geopend wanneer erop wordt geklikt. Dit attribuut kan een URL zijn of een object dat de naam van de route en eventuele parameters bevat.

### Router-view



De **router-view**-component is een ander onderdeel van de Vue Router, en is verantwoordelijk voor het renderen van de juiste component op basis van de huidige URL/route. De **router-view**-component wordt meestal gebruikt in de template tags van de component die fungeert als de hoofdpagina van je applicatie. (App.vue)

Wanneer een gebruiker naar een andere pagina navigeert via een **router-link**-component, zorgt de Vue Router ervoor dat het juiste component wordt gerenderd en weergegeven in de **router-view**-component.

Kortom, de Vue Router is een plugin voor Vue.js waarmee je client-side routing aan je applicatie kunt toevoegen. De **router-link**-component wordt gebruikt om links te maken die overeenkomen met gedefinieerde routes, en de **router-view**-component is verantwoordelijk voor het renderen van de juiste component op basis van de huidige URL.

# Ons eerste eigen component schrijven

Maak onder de map (directory) components een nieuwe vue file aan genaamd ProductCard.vue. Als je hiervoor de voorgedefinieerde template van webstorm gebruikt zou je file er nu zoals hieronder uit moeten zien.   
Refactor ook de HomeView.vue component van composition API naar options API.   
Dit doe je door de export default met de name van de component toe te voegen aan het script en door het setup-attribuut te verwijderen uit de script tag.  
Render vervolgens de ProductCard component in de template van de HomeView view.

Text

Description automatically generatedText

Description automatically generated

Je zou nu het h1 element van de ProductCard component moeten kunnen zien als je de / route, die de HomeView component renderd (zie router/index.js), moeten kunnen zien.

Text

Description automatically generated

# [Component data](https://vuejs.org/api/options-state.html#data)

De data() functie is een speciale eigenschap van Vue-componenten die wordt gebruikt om data te definiëren die in de template van het component wordt gebruikt. Deze functie retourneert een object dat alle gegevens bevat die in het component worden gebruikt en Vue houdt de wijzigingen van deze gegevens bij en zorgt ervoor dat de weergave automatisch wordt bijgewerkt wanneer er wijzigingen worden aangebracht.

Je kunt de **data()** functie gebruiken om variabelen te definiëren die de state van het component bijhouden, zoals een lijst met items die worden weergegeven in een winkelwagentje of een ingevoerd zoekwoord in een zoekveld. Je kunt deze variabelen vervolgens gebruiken in de template om de weergave van het component te bepalen.

Het is belangrijk om te onthouden dat de variabelen in de **data()** functie alleen in het huidige component beschikbaar zijn.

Een value kan als volgt in de component data worden opgeslagen en kan via de double mustaches worden weergegeven binnen een element in de template:

*in het <script> in de <template>*

Text

Description automatically generated with low confidenceText

Description automatically generated

#### Opdracht

Update de ProductCard component zodat die er zoals hieronder uit komt te zien. De tekst onder te image staat opgeslagen in de data van de component en wordt van daar uitgelezen.

Graphical user interface, text, application

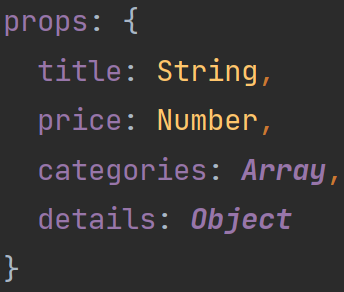
Description automatically generated

# [Component props](https://vuejs.org/guide/components/props.html#props)

Het props object is een manier om vanuit een parent component data door te geven aan een child component van die parent component. Het wordt gebruikt om bepaalde gegevens dynamisch door te kunnen geven aan de child component. Hierdoor kunnen componenten herbruikbaar gemaakt worden, aangezien de data die een component als input krijgt kan veranderen.

In plaats van gegevens direct in een component te definiëren, kunnen we deze gegevens doorgeven aan het component via props. Dit betekent dat we dezelfde component met verschillende gegevens kunnen gebruiken in verschillende delen van onze app.

Om een prop te definiëren, voeg je simpelweg een props object toe aan de component definitie (het export default object).   
Binnen dit props object gaan we elke prop die de component binnenkrijgt definiëren en we geven telkens mee wat voor datatype deze prop hoort te zijn. Dit is niet verplicht in vue maar zo lang als jullie studeren aan de PXL spreken we af dat het wel verplicht is om dit te doen.   
De reden dat we aan type-checking willen doen is omdat dit het opsporen van fouten aanzienlijk gemakkelijker maakt. Moest er een type mismatch zijn zouden we dat als een error in de console kunnen zien. Hierdoor weten we meteen dat onze code niet werkt omdat we iets verkeerd doorgeven vanuit ons parent component.

* voorbeeld van een props object met 4 verschillende props.*

Let wel op! Property namen in props en data moeten uniek zijn. Je mag dus niet een prop genaamd title hebben en tegelijkertijd een property in data hebben die ook title als naam heeft.  
Verwijder dus de title property uit data en we gaan deze waarde nu binnen laten komen vanuit de parent component via props. Price, categories en details hoef je voorlopig nog niet toe te voegen, deze staan hierboven enkel ter illustratie van verschillende datatypes.

Als aan het child component props is toegevoegd met daarbinnen een title property (zie hierboven) kunnen we die variabele gaan invullen in de parent component. Op de plaats waar we het child component renderen. Zo dus:

*parent component <template>: child Component <script>:*

Text

Description automatically generated

Props kunnen ook dynamisch doorgegeven worden. Als je niet een letterlijke string maar een expressie of een waarde uit een variabele van je component wilt doorgeven dan kan dat via de v-bind directive. Hierover in het volgende hoofdstuk meer. Het doorgeven van de prop zou er dan zo uit zien:

Text

Description automatically generated

*Text

Description automatically generatedbovenstaande template met de shorthand notatie van de v-bind*

# [Computed properties](https://vuejs.org/guide/essentials/computed.html#computed-properties)

Computed properties in Vue 3 zijn eigenschappen die worden berekend op basis van andere eigenschappen en reageren op wijzigingen in deze eigenschappen. Ze worden gedefinieerd als functies in de "computed" eigenschap van een component.

In dit voorbeeld hebben we een computed property genaamd "fullName", die is gebaseerd op de props "firstName" en "lastName". Telkens wanneer deze props worden gewijzigd, zal de computed eigenschap "fullName" opnieuw worden berekend.  
Text

Description automatically generatedHet is ook mogelijk om waardes uit je data() state te gebruiken binnen de berekening van je computed property.

# Directives

Bekijk voor elk van deze directives zeker de documentatie ook eens!   
De titels bevatten linken naar de officiële docs. de docs bevatten meer goede voorbeelden en extra uitleg die je kan helpen om de leerstof nog beter te begrijpen.

## [v-bind directive](https://vuejs.org/guide/essentials/class-and-style.html)

De v-bind directive is een Vue directive die wordt gebruikt om dynamisch attributen aan een HTML-element toe te voegen.

Met de v-bind, kun je de waarden van JavaScript-expressies koppelen aan de waarden van HTML-attributen.   
Dit betekent dat je de attributen van een HTML-element kunt wijzigen op basis van de waarden van je data-objecten in je Vue-component.

Je kan **v-bind** op twee manieren gebruiken: als shorthand, waarbij je alleen het **:** voorvoegsel gebruikt, of als een uitgeschreven vorm:

Text

Description automatically generated

V-bind kan ook gebruikt worden om conditioneel [class bindings](https://vuejs.org/guide/essentials/class-and-style.html#class-and-style-bindings) of [style bindings](https://vuejs.org/guide/essentials/class-and-style.html#binding-inline-styles) toe te passen. Dit wilt zeggen dat deze classes enkel toegepast zullen worden op het element als de expression erachter truthy is.   
De style property zal de waarde van de javascript expressie krijgen.

Text

Description automatically generated

## [Component methods](https://vuejs.org/guide/essentials/reactivity-fundamentals.html#declaring-methods)

Component-methoden in Vue zijn functies die zijn gedefinieerd in een component en die kunnen worden opgeroepen vanuit de template of andere methoden in hetzelfde component. Deze methoden kunnen worden gebruikt om bepaalde taken uit te voeren, zoals het manipuleren van gegevens in de component.  
**LET OP: Props die van de parent binnenkomen mogen nooit gewijzigd worden in het child component.**

Om een methode in een component te definiëren, voeg je deze toe aan de **methods** optie. De naam van de methode wordt een eigenschap van het component die kan worden opgeroepen vanuit de sjabloon of andere methoden. Bijvoorbeeld:

De event handler methodes die uitgevoerd moeten worden als er op de knoppen geklikt wordt komen onder **methods** te staan in de component. (zie hieronder)

## [v-on directive](https://vuejs.org/api/built-in-directives.html#v-on)

De v-on directive in Vue wordt gebruikt om event listeners toe te voegen aan elementen.

Dit kan worden gebruikt om te reageren op events zoals klikken, dubbelklikken, slepen, scrollen, toetsenbordinvoer, enzovoort.

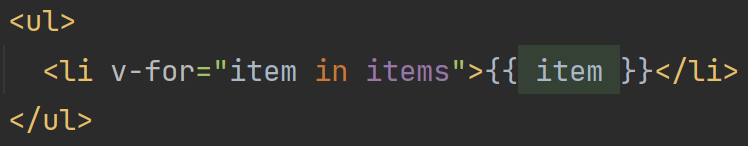
Om een **v-on** directive te gebruiken, moet je het event dat je wilt opvangen opgeven als de naam van de directive, gevolgd door de naam van de methode die moet worden opgeroepen wanneer het event optreedt.

In het onderstaande voorbeeld wordt er aan het click-event van de button een event listener toegevoegd. Dit kan ofwel met **v-on:click** of met de shorthand notatie: **@click** . Methods worden hier onder uitgelegd.

Text

Description automatically generated

## V-for directive

A picture containing logo

Description automatically generatedDe v-for directive wordt gebruikt om een stukje HTML code te herhalen voor elk element in een array of object. Het wordt gebruikt met de "in" notatie om aan te geven welke array of object gebruikt moet worden en welke variabele gebruikt moet worden om door de elementen heen te itereren.

Bovenstaande code zal een lijst van li elementen genereren voor elk item in de items array, waarbij de inhoud van het li element overeenkomt met het huidige item in de iteratie.

De bovenstaande v-for kan eigenlijk nog verbeterd worden door een key attribuut mee te geven. Het key attribuut wordt gebruikt om Vue te helpen onderscheid te maken tussen de verschillende elementen in de lijst en te optimaliseren hoe Vue de lijst weergeeft wanneer deze wordt bijgewerkt. Het key attribuut moet worden toegevoegd aan het element dat wordt herhaald en moet uniek zijn voor elk element in de lijst. Bijvoorbeeld:

Text

Description automatically generated

Text

Description automatically generated

## [v-model](https://vuejs.org/guide/components/v-model.html#component-v-model)

De v-model directive is een handige shorthand in Vue 3 voor het creëren van two-way data binding op form input elementen, zoals input, textarea en select elementen. Het stelt je in staat om de waarde van een input element direct te binden aan een eigenschap in de data van de component. Wanneer de waarde in het input element veranderd zal ook meteen de waarde in de data van de component worden gewijzigd.

Text

Description automatically generated

Neem dit voorbeeld zeker eens over en open je vue dev tools. Selecteer de component waar je two-way data binding gebruikt en typ in het inputveld. Je zou live de waarde in data moeten zien veranderen.

## v-if & v-else directive

De v-if en v-else directives worden gebruikt om elementen conditioneel te renderen op basis van een gegeven expressie.

De **v-if** directive voegt een element toe of verwijdert deze op basis van de truthyness (waarheidswaarde) van de expressie. Als de expressie **true** is, wordt het element aan de DOM toegevoegd. Als het **false** is, wordt het element uit de DOM verwijderd.

De **v-else** directive wordt samen met **v-if** gebruikt om een alternatief element aan de DOM toe te voegen wanneer de expressie **false** is. De **v-else** directive moet direct na een **v-if** directive volgen en heeft geen expressie.

Text

Description automatically generated

# [Emitting custom events](https://vuejs.org/guide/components/events.html)

De emit-methode in Vue 3 wordt gebruikt om events vanuit een kindcomponent naar een bovenliggend component te verzenden. Het stelt een kindcomponent in staat om een custom event te activeren en de bovenliggende component op de hoogte te stellen van die gebeurtenis, zodat het kan reageren.  
Via een custom event kun je ook gemakkelijk gegevens van een child component naar een parent component doorgeven, eigenlijk de omgekeerde dataflow van props dus.

Om **emit** te gebruiken, moet je eerst een custom event definiëren in het kindcomponent. Dit wordt gedaan met behulp van de **this.$emit**-methode, waarbij je de naam van het event en eventuele bijbehorende gegevens doorgeeft. Bijvoorbeeld:

*direct in de template:*  


*of ergens in het script, bv binnen een method. Hier wordt ook een parameter meegegeven met het event:*  


A picture containing text, font, screenshot

Description automatically generated

**In het bovenliggende component (parent)** kun je het aangepaste event ontvangen door een **v-on**-directive (of kortweg **@**-symbool) te gebruiken, gevolgd door de naam van het custom event en een functie die wordt uitgevoerd wanneer het event wordt geactiveerd. Binnen deze functie kun je de nodige logica uitvoeren. Bijvoorbeeld:



Merk hier op dat je in het child component de naam van je event camelCased doorgeeft en in de parent component je luisterd naar het event waar de naam in kebab-casing geschreven moet worden. Dit is omdat we ons moeten blijven houden aan de stijlregels van HTML en Javascript.

De event handler **handleEvent** staat gedefinieerd onder de **methods** van de parent component. Dus waar je met de v-on bent aan het luisteren naar het event op je child component.

A screen shot of a computer program

Description automatically generated with low confidence

# Dynamic image paths

Een dynamisch pad meegeven aan de image source gaat op de volgende manier:

A screen shot of a computer screen

Description automatically generated with low confidence

Als eerste moet het statische gedeelte van het pad tussen enkele aanhalingstekens geplaatst worden. En daarna wordt het dynamische gedeelte er achter geplakt met het concatinatieteken.

# Pinia

Pinia is een state management library speciaal ontworpen voor Vue 3-projecten. Het biedt een gestructureerde manier om de gedeelde state in je Vue-applicatie te beheren. Elk component in je applicatie kan in principe aan deze state moesten ze die nodig hebben.

In Vue-applicaties wordt state vaak gedeeld tussen meerdere componenten. Dit kan leiden tot complexe communicatie tussen componenten, prop drilling en onnodige her-rendering van componenten. Pinia biedt een elegante oplossing voor deze uitdagingen.

Met Pinia kun je een globale store maken die de state van de toepassing bevat. Deze store fungeert als de "single source of truth" voor je gegevens. Elke component kan zich vervolgens abonneren op de delen van de store die relevant zijn voor zijn functionaliteit. Hierdoor hebben componenten toegang tot de meest recente gegevens en kunnen ze automatisch opnieuw renderen wanneer de state verandert.

Wat Pinia onderscheidt, is het gebruik van TypeScript en Vue 3's reactiviteitssysteem. Met TypeScript kun je sterke typen definiëren voor je store en de gegevens die het bevat, waardoor je fouten kunt opsporen tijdens het ontwikkelproces. Het reactiviteitssysteem van Vue 3 zorgt ervoor dat je componenten automatisch opnieuw worden gerenderd wanneer de relevante gegevens in de store veranderen, zonder dat je handmatig moet bijwerken of gebeurtenissen moet uitstoten.

Pinia biedt ook enkele handige functies, zoals het ondersteunen van asynchrone acties en het mogelijk maken van devtools-integratie. Het is eenvoudig te gebruiken en heeft een kleine bundelgrootte, waardoor het een goede keuze is voor het beheren van de toestand in je Vue 3-projecten.

## De verschillende aspecten van een pinia store

Store-instantiatie:   
Om een Pinia-store te gebruiken, moet je deze eerst instantiëren met behulp van de defineStore-functie. Deze functie neemt een naam en een configuratieobject als parameters en retourneert een storeklasse die je kunt gebruiken om een instantie van de store te maken.

State:   
Het state-object in een Pinia-store bevat de gegevens die je wilt delen tussen componenten. Het wordt gedefinieerd als een JavaScript-object met de gewenste eigenschappen en hun beginwaarden. Pinia maakt gebruik van Vue 3's reactiviteitssysteem, dus elke wijziging in de state zal automatisch de bijbehorende componenten opnieuw renderen.

Getters:  
 Getters in een Pinia-store zijn computed properties die zijn afgeleid van de state. Ze bieden een manier om dynamische waarden te genereren op basis van de huidige state. Getters worden gedefinieerd als methoden in de storeklasse en kunnen worden gebruikt in componenten om specifieke waarden uit de store op te halen.

Actions:   
Actions vertegenwoordigen bewerkingen die kunnen worden uitgevoerd op de store. Ze worden gedefinieerd als methoden in de storeklasse en kunnen asynchroon zijn. Actions worden opgeroepen vanuit componenten en kunnen de staat van de store wijzigen. Ze bieden een gestructureerde manier om bewerkingen uit te voeren en zorgen ervoor dat de state consistent blijft.

Mutations: (niet op het examen)  
Mutations zijn vergelijkbaar met actions, maar ze worden gebruikt om de state synchroon te wijzigen. In tegenstelling tot actions, die asynchroon kunnen zijn, moeten mutaties directe en onmiddellijke wijzigingen aan de toestand doorvoeren. Mutaties worden opgeroepen vanuit actions en bieden een manier om de toestand te wijzigen terwijl je de wijzigingen traceerbaar houdt.

Modules: (niet op het examen)  
Als je applicatie een complexe state heeft, kun je de store modulair maken met behulp van modules. Een module is een afzonderlijke instantie van de storeklasse met zijn eigen state, getters, actions en mutations. Modules kunnen worden genest, waardoor je een hiërarchische structuur kunt creëren die overeenkomt met de structuur van je toepassing.

Plugins: (niet op het examen)  
Pinia biedt de mogelijkheid om plugins toe te voegen aan je store. Plugins zijn functies die worden uitgevoerd tijdens de instantiatie van de store en kunnen worden gebruikt om extra functionaliteit toe te voegen, zoals het toevoegen van externe libraries of het instellen van vooraf gedefinieerde gegevens.

## Een pinia store definiëren

Allereerst is het belangrijk om te begrijpen dat een Pinia store gedefinieerd wordt door de **defineStore** methode. En dat deze methode als eerste parameter een naam of id voor de store moet krijgen.

Deze naam, ook wel id genoemd, is noodzakelijk en wordt door Pinia gebruikt om de store te verbinden met de devtools. Het benoemen van de geretourneerde functie als use... is een conventie in composables om het gebruik ervan idiomatisch te maken.

**defineStore()** accepteert twee verschillende waarden als tweede argument: een Setup-functie **OF** een Options-object. Hier wilt dit dus ook weer zeggen of je de store via de **composition API** of de **options API** gaat schrijven.

A screen shot of a computer code

Description automatically generated with low confidence*Een voorbeeld van de options API*

## State, actions, getters

A screen shot of a computer program

Description automatically generated with low confidence

In dit voorbeeld is al wat basic functionaliteit aan de Pinia store toegevoegd. Eigenlijk bevat pinia zo goed als geen nieuwe leerstof aangezien je al een soortgelijke manier van werken gewoon bent van je componenten te schrijven. In Pinia krijgen de properties gewoon een andere naam.

Even overlopen:

|  |  |
| --- | --- |
| VUE COMPONENT | PINIA STORE |
| data( ) | **state( )** |
| computed | **getters** |
| methods | **actions** |

## De Pinia store aanspreken in componenten

A screen shot of a computer program

Description automatically generated with low confidence

# Bijlages

### App.vue options api

<template>  
 <header>  
 <img alt="Vue logo" class="logo" src="@/assets/logo.svg" width="125" height="125" />  
  
 <div class="wrapper">  
 <HelloWorld msg="You did it!" />  
  
 <nav>  
 <router-link to="/">Home</router-link>  
 <router-link to="/about">About</router-link>  
 </nav>  
 </div>  
 </header>  
  
 <router-view />  
</template>  
  
<script>  
import HelloWorld from './components/HelloWorld.vue'  
  
export default {  
 components: {  
 HelloWorld,  
 },  
}  
</script>  
  
  
<style scoped>  
header {  
 line-height: 1.5;  
 max-height: 100vh;  
}  
  
.logo {  
 display: block;  
 margin: 0 auto 2rem;  
}  
  
nav {  
 width: 100%;  
 font-size: 12px;  
 text-align: center;  
 margin-top: 2rem;  
}  
  
nav a.router-link-exact-active {  
 color: var(--color-text);  
}  
  
nav a.router-link-exact-active:hover {  
 background-color: transparent;  
}  
  
nav a {  
 display: inline-block;  
 padding: 0 1rem;  
 border-left: 1px solid var(--color-border);  
}  
  
nav a:first-of-type {  
 border: 0;  
}  
  
@media (min-width: 1024px) {  
 header {  
 display: flex;  
 place-items: center;  
 padding-right: calc(var(--section-gap) / 2);  
 }  
  
 .logo {  
 margin: 0 2rem 0 0;  
 }  
  
 header .wrapper {  
 display: flex;  
 place-items: flex-start;  
 flex-wrap: wrap;  
 }  
  
 nav {  
 text-align: left;  
 margin-left: -1rem;  
 font-size: 1rem;  
  
 padding: 1rem 0;  
 margin-top: 1rem;  
 }  
}  
</style>