FRONT MANIA CONFERENCE 2022

October 5th

FRONTMANIA MAGAZINE SPECIAL

Timetable

Floorplan

Articles

FRONTMANIA
welcome to the jungle!

Mathijs de Groot
Board Member
Frontmania
Divotion

Alwin Captijn
Board Member
Frontmania
Pancompany

Welcome

Today is all about exploring, seeing new tech, hearing fresh ideas and being inspired in the field of our beloved Frontend development.

At today's conference, we strive to provide you with the very best DX (Developer Xperience). A conference created for and by Frontend developers.
On behalf of the Frontmania board, we would like to give you all a warm welcome to the Frontend community. We want to create a community for Frontend developers, where we can meet, learn and have fun together.
First of all, we would like to thank our sponsors as they make this community and conference possible and accessible for all. We want to thank our sponsors Rabobank, iO, Sauce Labs, ABN AMRO, AWS, Ordina, Divotion & Sogeti. We're especially excited that they not only provide their support, but also bring great content to the table such as the sessions Container Queries: The next step towards a truly modular CSS and Modernize your APIs with GraphQL and even a workshop by ING on LitElements.

let us introduce ourselves

Mathijs de Groot, I've worked as a software developer since 2005 in various roles on both small and very large projects contributing to the backend, frontend, cloud and all things DevOps. Currently, I dedicate my efforts to create a warm working environment for passionate Frontend developers at Divotion. In my spare time, I like to explore the world in both active and passive ways. I love travelling with my backpack, exploring new places and relaxing on deserted sandy beaches. This combines perfectly with my love for water sports such as (wind)surfing, diving, kayaking, or really all activities in and around the water.
Alwin Captijn, working at PanCompany since 2012. Started as a consultant and made the transition to Business Manager Frontend, UX and Mobile Development. How do colleagues describe me? That smiling extrovert guy with his crazy shoes who likes to be at our parties. On the other hand, hard working to grow the competence and keep that club-feeling. The Dungeon Master for the PanCompany RolePlay Group, but also present at our knowledge meet-ups. A little more about myself: huge passion for sports, especially fitness & mountain biking, Dungeons & Dragons, riding my motorcycle and also exploring cool places around the world.
We've both become board members of Frontmania because we believe in the power of communities, because 'The whole is greater than the sum of its parts' – Aristotle. The other board members will be presenting themselves in the upcoming Frontmania Magazine.

program

The program committee has worked hard to create a program with 5 parallel tracks, with over 25 sessions presented by 30+ speakers from all around the world. We'd like to thank the members of the program committee (Maurice de Beijer, Lucien Immink, Frank Merema, Wim Selles & Martijn van der Wijst) for their efforts in creating an amazing and balanced program. We'd love to hear your feedback on the program. You can send us your feedback using the #frontmania hashtag on Twitter, through the evaluation after the event.

thanks

Where would we be without the great help of the organising team at Reshift, the sponsors, the program committee, all the volunteers, the speakers and of course all of you! Especially after such a long time, due to COVID, it's an honour to get back together and be able to write this "thank you" note.
At the Frontmania conference, you will recognize the Frontmania board members by our Frontmania shirts. If you happen to bump into us, please don't hesitate to ask us any questions you might have, to give us your feedback about the event or just have a chat.
We hope that you will have a great day where you gain a lot of knowledge, meet new people and, most importantly, will have lots of fun!

sogeti
Part of Capgemini

We make technology work

Wij zijn een technologiebedrijf in hart en nieren waar jij in staat wordt gesteld het beste uit jezelf naar boven te halen. Of je nou een ervaren of beginnende front-end developer bent, jouw talent kan altijd verder groeien. Kom jij onze community van front-end professionals versterken? Kijk op:

www.werkenbijsogeti.nl

Daarom Sogeti:

- *Werken voor topmerken in Nederland*
- *Breed opleidingsprogramma*
- *Ruimte voor persoonlijke ontwikkeling*
- *Kennisdelen bij Frontend Lightning Talks*
- *Open en collegiale sfeer*
- *Technische kwaliteit op nummer 1*
- *Ruimte voor initiatieven en meer dan alleen developen*

Software Bastards

Wij "Bastards" hebben een grote passie voor software development. Met de beste talenten uit de markt, creëren en implementeren wij technische oplossingen, die direct waarde toevoegen aan diverse organisaties.

Wij sturen onze Bastards graag op missie voor de meest uiteenlopende projecten! Zij deinzen niet terug voor moeilijke greenfield projecten, een state-of-the-art platform of het bouwen van een grootschalig beveiligingsplatform. Met onze kennis en capaciteiten kunnen wij meedenken, adviseren over de oplossing en direct invulling geven aan de opdracht. Wij brengen de juiste Bastard, op de juiste plaats en zorgen dat iedere missie wordt volbracht!
Wij zijn de altijd up-to-date-special forces die niet alleen vooruit, maar ook 360° om ons heen kijken. Wij willen bijdragen aan het succes van onze klanten door middel van front- en back-end coding en software development. Met onze transparante, geïntegreerde aanpak en hands-on mentaliteit zijn wij de geschikte partner voor zowel grotere organisaties, zoals banken en overheidsinstellingen, maar ook voor bedrijven in het MKB, Tech startups en scale-ups. Wij lopen voorop in digitale transformatie en linken de juiste specialistische kennis aan onze klanten. Vanuit onze passie voor software development en ons Bastards-DNA werken wij aan front- en back-end oplossingen die waarde toevoegen aan onze klanten.
Niets is zo blijvend als verandering. "Slimmer, efficiënt en data" zijn de kernwoorden van deze tijd. Wij zijn dan ook de special forces die altijd een stapje verder willen en zijn in de markt van digitale technologie. Samen met onze klanten creëren we digitale transformatieprocessen die voldoen aan de veeleisende verwachtingen van vandaag. Een duidelijke strategie voor het gebruik en analyse van data en de inzet van de juiste technologie zijn dan ook de ingrediënten voor een gezamenlijk succes.
Onze cultuur doet denken uit die van het squadron van hoofdrolspeler Tom Cruise in de film Top Gun (een film over straaljagerpiloten, wie kent hem niet?). Kameraadschap, vertrouwen, ambitie, eerlijkheid, vrijheid, verantwoordelijkheid en eigenzinnigheid kenmerken de cultuur bij Software Bastards.
Wij geloven in het motto 'Alleen ga je sneller, maar samen kom je verder'. We zijn een team van eigenzinnige, vakbekwame individuen die altijd op elkaar kunnen terugvallen. We helpen elkaar altijd waar nodig, maar we kijken niet constant over elkaars schouder mee. We nemen onze eigen verantwoordelijkheid om onze projecten tot een goed einde te brengen en maken gebruik van elkaars expertise waar nodig. Altijd met het gezamenlijke doel om te excelleren.
Bij Software Bastards zijn we een team met allemaal unieke individuen. Wij zijn trots op onze cultuur en willen waarborgen wat we hebben. Met elkaar willen we altijd winnen en het beste zijn in ons vak. We staan we voor elkaar klaar en brengen niet alleen ons zelf maar ook de ander naar een hoger niveau. Bij Software Bastards kun je zijn wie je bent en worden wie jij wilt zijn!

Flight SWB-0510.

Flight: SWB-0510 stand 05

VRIJHEID VAKVOLWASSENHEID java fullstack top gun node react wingman angular music backend beach party frontend vue maverick fullstack top gun vue php maverick beach wingman php VERANTWOORDELIJKHEID backend vertrouwen wingman EIGENZINNIG java angular react maverick vrijheid party java node frontend music php fullstack erecode fullstack verantwoordelijkheid vue erecode music top gun broederschap java react wingman backend AMBITIE erecode node angular BROEDERSCHAP backend vakvolwassenheid party party beach maverick react erecode node vue beach frontend music EERLIJKHEID top gun ambitie angular eerlijkheid frontend eigenzinnig php VERTROUWEN

Vlieg met ons mee bij de stand met die tent & VR-simulatoren!


TIKKIE WON'T BE INNOVATIVE FOREVER
#STARTWITHUS
After successfully launching apps like Tikkie and Grip, it would be easy to say "Okay, we made it". But that's not how innovation works. Innovation is about constantly challenging yourself. Daring to acknowledge that something can always be better. Must be better. Because it's this mindset that makes the difference. And to make a difference we need you. Employees that will continue to motivate others, and us.
www.workingatabnamro.com/FrontMania
ABN·AMRO

gebruik jij al een micro frontend?

een terugblik op het ontstaan van deze techniek.

De afgelopen paar jaar heb ik steeds vaker mogen werken aan een micro frontend, althans zo werden ze genoemd door de enterprise organisaties waar ik gewerkt heb. Als ontwikkelaar merkte ik echter niet heel veel verschil in mijn alledaagse werkzaamheden. Ik werkte nog steeds met een framework of library, zoals Angular, React of Vue en ik schreef nog steeds HTML, CSS en Java- of Typescript. Dus wat maakte het werken aan deze applicaties dan zó anders waardoor het een andere naam had gekregen? Waar komt de term vandaan en hoe is deze techniek eigenlijk ontstaan?

De term 'micro frontend' kwam in de Thoughtworks Technology Radar in 2016 voor het eerst naar boven in de 'assess'-categorie. Dit is een categorie waarin er wordt gekeken of een bepaalde techniek of oplossing mogelijk interessant is voor ontwikkelaars en bedrijven. Het trok grote lijnen uit de destijds al geadopteerde microservice architectuur, waarbij een applicatie wordt opgeknipt in kleinere applicaties.

Hierdoor is het eenvoudiger om de verschillende functionaliteiten te onderhouden of uit te breiden en is het zelfs mogelijk om meerdere teams tegelijkertijd aan verschillende functionaliteiten te laten werken, iets dat een stuk lastiger was om voor elkaar te krijgen met de monoliet oplossing die voorheen redelijk de standaard was.

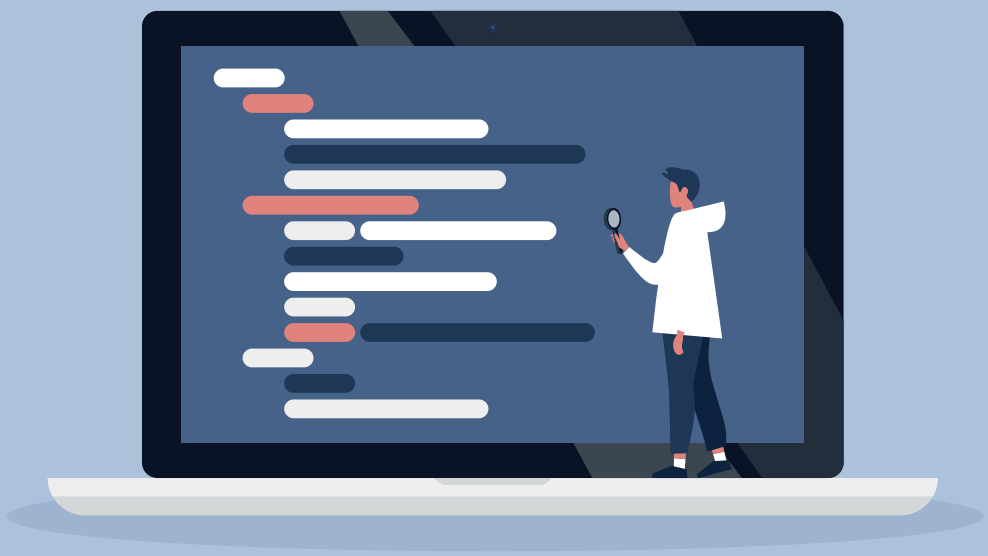
Deze splitsing in functionaliteiten kwam niet geheel uit het niets. In voorgaande jaren was het al een standaard geworden om in elk geval het UI-aspect uit verschillende enterprise-applicaties te halen en deze apart op te zetten. Voorbij waren de tijden van het integreren van je frontend in de PHP- of JSP-bestanden aan de backend, en voor ons frontend-ontwikkelaars begon toen het Web App of SPA-tijdperk waar we met behulp van REST APIs fantastische webapplicaties konden neerzetten!

Maar met deze nieuwe manier van denken, kwam ook een nieuw risico naar boven: de wel bekende 'scope-creep'. Steeds vaker werden er functionaliteiten die we normaliter in de backend bouwde, ook in de frontend verwacht. Denk hierbij aan complexere dingen, zoals de benodigde formules en berekeningen voor bepaalde grafieken of tabellen. Maar ook aan de simpelere dingen, zoals validatie op gebruikersinvoer voordat het naar de backend wordt verstuurd.

In de frontend gingen we steeds vaker richting een probleem waar de backend al een oplossing voor had gevonden. De webappli-

BIO

***Dymion Fritz** is een ervaren Software Engineer met een liefde voor architectuur en nieuwe technieken. Hij begon in 2010 als Android en iOS-developer tot hij in 2014 de switch maakte naar Fullstack developer met een focus op frontend. Sindsdien heeft hij meerdere enterprise organisaties geholpen met het ontwikkelen van hun applicaties en bekleedt hij sinds een paar jaar de rol van CTO. Programmeren is zijn passie en zijn rol als Software Engineer zal altijd voorop staan in zijn doen en laten. Want van een mooie code oplossing wordt elke developer gelukkig.*



caties die we bouwden neigde steeds meer richting een monoliet die de informatie van verschillende microservices moest ophalen en verwerken. Maar als de backend hier al een oplossing voor had gevonden in de vorm van de microservice architectuur, zou dit mogelijk ook een oplossing kunnen zijn voor ons monoliet probleem in de frontend-applicaties.

Een van de eerste signalen richting een microservice architectuur die wij in de frontend zagen, was de toename in gebruik van eigen ontwikkelde en beheerde library's. Deze library's omvatte specifieke functionaliteiten die waren losgetrokken van de applicatie en die (vaak met behulp van een private NPM-registry) weer werden afgenomen door de frontend-applicatie waar de functionaliteit oorspronkelijk in verwerkt zat.

IN DE FRONTEND GINGEN WE STEEDS VAKER RICHTING EEN PROBLEEM WAAR DE BACKEND AL EEN OPLOSSING VOOR HAD GEVONDEN.

Op deze manier was het mogelijk om meerdere teams aan de verschillende functionaliteiten van een frontend-applicatie te laten werken zonder dat deze teams elkaar in de weg zaten.

De gedeelde functionaliteiten varieerden van specifieke applicatielogica tot volledige UI-componenten die voor een uniforme interactie zorgde, maar uiteindelijk zat het vaak allemaal nog in één frontend-applicatie. De logische volgende stap was dan ook proberen om deze ene frontend-applicatie op te knippen in verschillende losse applicaties, zodat deze ook individueel ontwikkeld, beheerd en gereleased konden worden.

Vanaf dit punt werd het concept van een micro-app geïntroduceerd. Een applicatie die één specifieke functionaliteit of domein van een grotere applicatie omvat. Denk bijvoorbeeld aan een verzekerings-applicatie, waarin je meerdere soorten verzekeringen kan beheren of afnemen. Waar je voorheen in één grote web-applicatie zowel het overzicht en de auto- en woonverzekering pagina's had, kon je nu het overzicht en elke losse verzekering als individuele webapplicatie opzetten en gebruiken. Qua architectuur voor de frontend zaten we na deze laatste stap op vrijwel hetzelfde niveau als de microservice architectuur die al langer in de backend werd toegepast.

Deze signalen en veranderingen zagen wij voor het eerst rond het begin van 2018, ongeveer dezelfde tijd waarin de term 'micro frontend' verschoof van de 'assess'-categorie naar de 'trial'-categorie in de Thoughtworks Technology Radar. In de 'trial'-categorie wordt het aangeraden aan enterprise organisaties om te experimenteren met een bepaalde techniek of oplossing, zoals wij in dit geval hebben kunnen zien met de micro frontend-techniek.

Inmiddels is het 2022 en zijn de beschikbare technologieën en tools dusdanig gegroeid, dat het eenvoudiger is dan ooit om te beginnen met de micro frontend-techniek. Dankzij de adoptie van patterns, zoals 'Domain Driven Design' en de opkomst van mono-repos, is het steeds duidelijker waar de scheiding in functionaliteit zit en hoe deze gefaciliteerd kan worden voor de verschillende frontend-applicaties.

Er zijn, zoals altijd in de developerwereld, verschillende manieren om met micro frontends om te gaan. Denk aan het gebruik van losse applicaties die elk individueel van elkaar functioneren, of het samenvoegen van verschillende applicaties met behulp van een app-shell voor overkoepelende communicatie. De mogelijkheden zijn eindeloos.

Inmiddels komt de term 'micro frontend' ook niet meer voor in de Thoughtworks Technology Radar, dit omdat de term inmiddels is geadopteerd en een huis-tuin-en-keuken begrip is geworden voor de meeste ontwikkelaars. Dus, gebruik jij al een micro frontend?

Sustainable frontend development: hoe maak je de juiste keuze?

There is an app for that! IT als de oplossing voor elk probleem. Dat is de mindset waardoor developers die leven voor het maken van impact elke dag met frisse energie opstaan. Kijkend naar duurzaamheid kom je dan uit op IT for good, Green IT en het allesomvattende Sustainable IT.

Tegelijkertijd neemt jaarlijks het energieverbruik van digitale technologieën sterk toe. Keuzes die we tijdens het ontwikkelen maken blijken onbewust van grote invloed op het energieverbruik, de CO2 uitstoot en gebruik van schaarse grondstoffen.

over-engineering

Jaren geleden ontwikkelde ik met Angular en Bootstrap mijn persoonlijke website emielkwakkel.nl. Enigszins over-engineered, maar hoeveel uitstoot kan een kleine persoonlijke website veroorzaken? Recent schrok ik van de resultaten van websitecarbon.com, een website die een benadering kan geven van de uitstoot van je website op basis van tal van factoren. Mijn website bleek vervuilender dan 93% van de geteste websites met vijf gram CO_2 uitstoot per page view. Tienduizend page views bleek gelijk te staan aan het energieverbruik waarmee een elektrische auto negenduizend kilometer kan rijden. Daarmee kan je twee keer op en neer naar Griekenland rijden!
Nu maak ik mij geen illusies over duizenden bezoekers op mijn persoonlijke website. Wel bouw ik aan front-end applicaties voor bijvoorbeeld banken, verzekeraars en overheden, goed voor miljoenen pageviews en intensief gebruik. Welke impact maken de keuzes die ik daar maak? En wordt in de architectuur rekening gehouden met de best practices?
Duurzaamheid heb ik altijd een belangrijk onderwerp gevonden. Tegelijk heb ik het tot voor kort onbewust niet meegenomen als harde non-functional requirement. Bij het opzetten van mijn server keek ik naar schaalbaarheid, performance en redundantie. Kritisch gezien kan je de vraag stellen: hoeveel cores hebben een aantal kleine Node based websites echt nodig? En is een cloud backup van een standaard ingerichte server instance nuttig wanneer alle eigen broncode op GitHub staat?

tijd voor een disclaimer

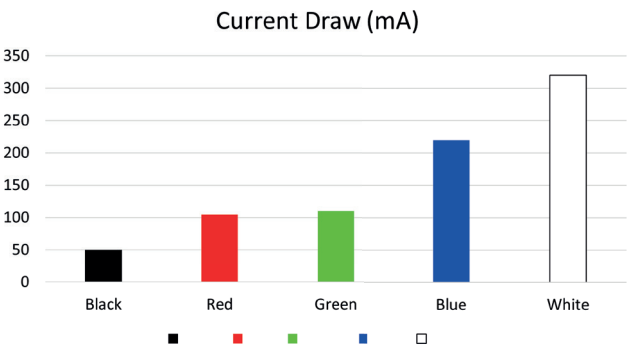
De strekking van onderstaand verhaal is het belang om op basis van de use case de juiste architectuur te kiezen en uitstoot door over-engineering te voorkomen. De combinatie Angular, Single Page Application met Bootstrap kan voor een dynamische Enterprise applicatie, mits geoptimaliseerd, een duurzame keuze zijn.

BIO

***Emiel Kwakkel** is de Lead Frontend van Sogeti en is momenteel ingezet bij de Politie. Intern ondersteund Emiel de frontend community en is hij betrokken als coach en docent bij het ontwikkelen van talenten in het vak.*

tijd voor verandering

Een nieuwe lichtgewicht VPS bleek evengoed instaat mijn website snel te serveren en leverde een maandelijkse besparing op van 24 euro. Ook mijn website ging op de schop.
Voor mijn persoonlijke website kwam ik uit op de Vue 3 & Nuxt 3 combinatie met Static Site Generation (SSG) en Tailwind voor de styling. De vernieuwde Vue Virtual DOM zorgt voor snellere rendering gecombineerd met verminderd geheugen gebruik. Gecombineerd met Nuxt wordt het framework grotendeels tree-shakable waardoor bundle sizes verkleinen. Daarnaast biedt Tailwind met de utility classes veel mogelijkheden om zelf de website te stylen, zonder de overhead van ongebruikte components.



Bron: https://www.wholegraindigital.com/blog/dark-colour-web-design/

Ook de styling bleek van invloed op het energieverbruik van de applicatie. Wist je dat een dark mode significant minder energie gebruikt? Begin 2000, in de tijd van de ouderwetse CRT-monitoren, ging de website van Google op zwart tijdens Earth Day. Met de stap naar LCD-schermen (single backlight, always on) verviel echter het voordeel. Sinds de opkomst van OLED-schermen is de dark mode weer terug van weggeweest. Uit onderzoek van Google bleek dat Google Maps in dark mode op mobiele devices tot 63% minder energie verbruikte voor het scherm. Win-win voor iedere developer met voorliefde voor dark mode!

het resultaat

Met de nieuwe website live werd het tijd voor een nieuwe scan bij websitecarbon.com. De vijf gram CO_2 per page view is nu gereduceerd naar 0,06 gram en 10.000 page views staan nu gelijk aan een autorit van 112 kilometer! Door de gereduceerde bundle size gecombineerd met de Static Site Generation zijn de laadtijden verbeterd. De GitHub workflow deployed in één minuut de site naar productie en de kosten voor de server zijn met 500% gedaald.

in de echte wereld

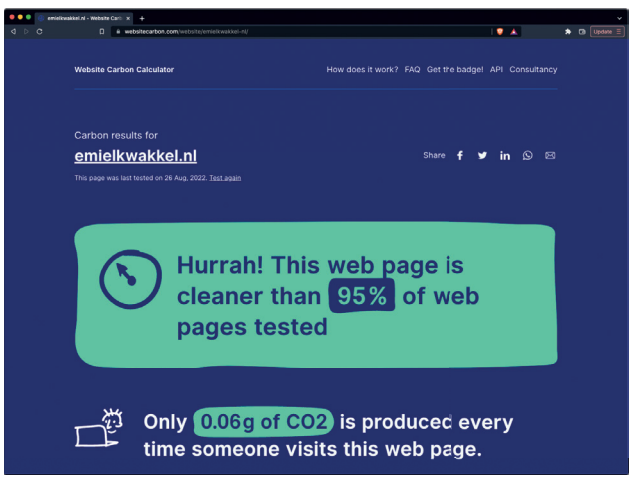
Met het vergroten van de schaal van het project ontstaan ook andere uitdagingen. Aan de development kant groeit in rap tempo het aantal testen. Gecombineerd met extra quality gates, linting en uitgebreidere deployment scripting groeit daarmee ook de doorlooptijd van de pipeline. Zowel een microfrontend als monorepository architectuur kunnen uitkomst bieden. Uitgangspunt is dat geautomatiseerd in kaart gebracht kan worden wat de impact is van een wijziging. Hiermee kan op basis van de dependency tree enkel de testen, builds en deployments worden uitgevoerd die geraakt worden.

Voorbeeld voor een dergelijke opzet is de KickstartGreen applicatie, gemaakt voor een fictieve bank die naast bankieren klanten helpt met het planten van bomen. De architectuur laat zich omschrijven als een Angular monorepository met NX, opgedeeld in projecten volgens het Domain Driven Design principe van Manfred Stayer. In deze opzet kan op applicatie, domein en library niveau worden ingezoomed. Een affected analyse highlight de libraries binnen de applicatie die geraakt worden door een wijziging.

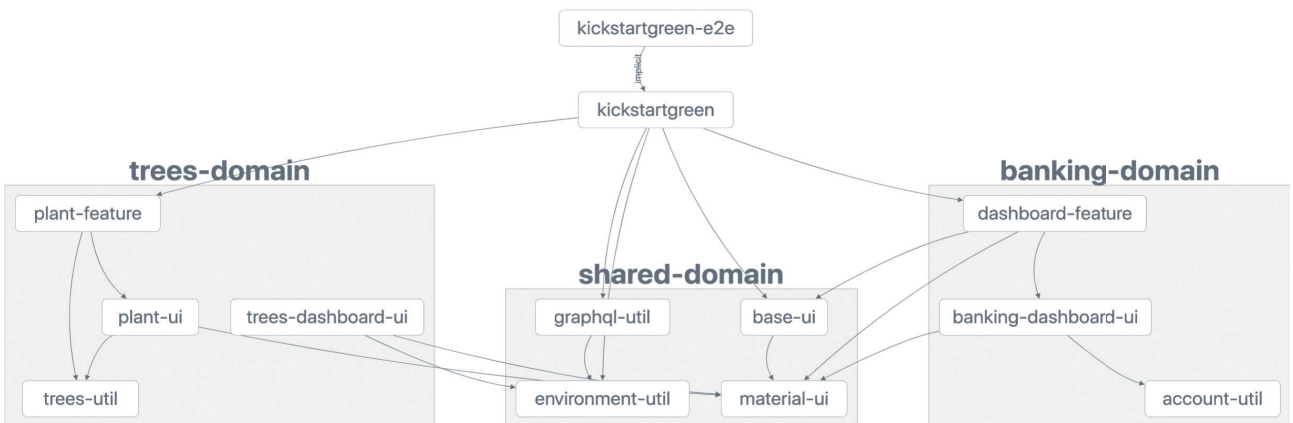
Bij het opsplitsen van de applicatie in domeinen en libraries moet goed nagedacht worden over de hiërarchie. Afhankelijkheden mogen enkel naar beneden lopen. Zou een utility library bijvoorbeeld een applicatie importeren dan zou een wijziging aan die utility betekenen dat ook de gehele applicatie als affected wordt aangemerkt, en daarmee ook alle onderliggende packages.

De lijst met mogelijke optimalisaties is te groot voor dit artikel. In basis blijkt bewustwording en het benoemen van duurzaamheid als harde eis de enige manier om afgewogen en meetbare keuzes te maken. Werk je voor een organisatie zonder harde duurzaamheidsdoelstellingen? Bekijk dan hoe een duurzaam voorstel ook kosten besparen, de onderhoudbaarheid vergroten, of bijvoorbeeld de time-to-market kunnen verbeteren. Win-win dus!

Alle projecten genoemd in dit artikel staan op mijn persoonlijke GitHub [1].



Figuur 1: KickstartGreen Dependency Graph in kaart gebracht met NX



Referenties

1. GitHub: https://github.com/emielkwakkel

How to automatically reset mocks and restore spies in Jest

When I used jest for the first time for unit testing, it struck me that function mocks and spies were not automatically reset / restored before each unit test execution. This was quite unexpected to me, especially when you are used to automatic reset / restore functionality of Jasmine.

BIO

Emil van Galen is Frontend Tech Lead at Divotion.

When using Jest it seemed to be a common approach to manually invoke **jest.resetAllMocks()** or **jest.restoreAllMocks()** inside a **beforeEach(..)** or **afterEach(..)**. Like this:

```
const myMock = jest.fn();

describe('..', () => {
beforeEach(() => {
          jest.resetAllMocks();
   });
```

Configuring Jest to automatically reset / restore mocks and spies

As it seemed, it turned out Jest can be configured to do an automatic reset / restore before executing each unit test spec. You can simply use these settings in the configuration of Jest:

- **"clearMocks": true**: resets all the mocks usage data, but keeps the behaviour (e.g. return value) of the mocks. It's effectively the same as: **beforeEach(() => { jest.clearAllMocks(); });**
- **"resetMocks": true** : same as "clearMocks": true but also resets the behaviour of the mocks. It's effectively the same as: **beforeEach(() => { jest.resetAllMocks(); });**
- **"restoreMocks": true** : restores methods that were spied using **jest.spyOn(..)** to its original method. This flag is independent of the clearMocks / resetMocks flags. Is effective the same as: **beforeEach(() => { jest.restoreAllMocks(); });**

The settings described above can be placed either:

- inside the **"jest""** entry in the **package.json**
- or in a Jest configuration file (typically called **jest.config.js**)

I personally configured Jest by placing listing 1 in **package.json** :

```
{
..
"jest": {
..
"resetMocks": true,
"restoreMocks": true
}
}
```

Listing 1

NOTE: when using Create React App the only officially supported way to configure Jest is through the package.json file.

How to properly spy on existing methods

Once in a while you need to replace a method of an existing (global) object with a Jest spy. For instance to be able to test if and how a method was called, or even to temporarily replace the behaviour of the method (e.g. returning a mocked return value). The poor man's way to spy on a method would to simply monkey-patch an object, by simply assigning the result of **jest.fn(..)** However, when manually replacing an existing method with a **jest.fn(..)**, you're also responsible to restore the original method. Instead, it's much better to use **jest.spyOn(..)**, in which case Jest automatically resets the spy when **"restoreMocks": true** is configured. Since **"restoreMocks": true** automatically restores a spy prior to executing each unit test spec (and prior to any custom beforeEach(..)), it's best to only use **jest.spyOn(..)** inside either:

- a **beforeEach(..)**
- a unit test spec

Whereas the following usage of **jest.spyOn(..)** will give issues: jest.spyOn(window, 'open').

```
describe('...', () => {
  it('...', () => { // you will get a fresh spy here
    // ..
    expect(window.open).toHaveBeenCalled();
  }

  it('...', () => { // X window.open is no longer a spy
    // ..
    expect(window.open).toHaveBeenCalled();
```

How to prevent method overrides with jest.fn()

To guard your codebase against the overriding a method by reassigning it with **jest.fn(..)**, you could configure the ESLint linter to use the prefer-spy-on rule [1].

Reference

1. https://github.com/jest-community/eslint-plugin-jest/blob/main/docs/rules/prefer-spy-on.md#suggest-using-jestspyon-prefer-spy-on

Standalone components

In release 14 of Angular, a new feature was introduced that may prove to be a game-changer in the way we structure Angular applications. With standalone components, we do not longer need to use the ngModule to declare a component in Angular. The component can declare itself and all its dependencies and exports. The introduction of standalone components is still in an experimental phase but it is probably the greatest change in Angular since the introduction of Ivy. What does this change entail and what impact will it have on developing in Angular?

The introduction of standalone components doesn't mean the end of the Angular module (ngModule). The concept is also coined as 'optional modules', from which we can take that modules are from now on optional. The two concepts are interchangeable: a standalone component can import from a module and vice versa. This will make it easy to introduce the new concept in an existing codebase.

What does a standalone component look like?

For a component to be treated as standalone, the decorator is extended with the new – optional - property 'standalone'. If set to true, Angular will treat the component as standalone. All properties we know for the ngModule also apply to a standalone component. To demonstrate this feature: In the following code snippet a standalone component is declared. The components' template has a dependency on another standalone component and on a component declared in an ngModule. The template also contains a condition in the form of an `ngIf`. In order to be able to use the `ngIf` directive, we need to import Angulars' common module as well. Finally, the component exports itself so it can be used by other components or modules. **listing 1**.

> ONE OF THE ARGUMENTS WHY THE ANGULAR TEAM INTRODUCED STANDALONE COMPONENTS IS TO MAKE IT EASIER FOR NEW DEVELOPERS TO ADAPT TO THE FRAMEWORK

What about bootstrapping and routing?

An Angular app bootstraps through a module at root level and its routes are also declared through modules. Does this mean we still need to use at least a few modules in our project? The answer is: no. Angular has created a solution to allow main.ts to load a component instead of a module. And since the introduction of Ivy, it was already possible to lazy-load components inside another component. So with these new introductions, it is fairly easy to set up a small Angular app with, theoretically, just one component, a main.ts and an index.html.

Flattening the learning curve

One of the arguments why the Angular team introduced standalone components is to make it easier for new developers to adapt to the framework. Angular has always been somewhat intimidating to developers and the ngModule is without doubt one of the reasons why.

Learning Angular while using standalone components, it should be easier to understand the concepts, while developing Angular should become less cumbersome. Dependencies are often 'implicitly' imported through the module and can easily be forgotten when importing a component. For many developers, it is not clear when to use modules at all and in one codebase you can find

L1

```
import { CommonModule } from '@angular/common';
import { Component } from '@angular/core';
import { ClassicComponentModule} from '@classic-component.module";
import { ChildStandaloneComponent } from './child-standalone.component';


@Component({
  selector: 'sa-component',
  standalone: true,
  imports: [
      CommonModule,
      AnotherStandaloneComponent,
      ClassicComponentModule
  ],
  exports: [StandaloneComponent]
  template:
  `<div *ngIf="foo">
    <sa-another-standalone></sa-another-standalone>
    <classic-component></classic-component>
  </div>`
})
export class StandAloneComponent  {
  foo = true;
}
```

libraries using a module for each component, while in others there is just one big module handling all. This kind of confusion should no longer exist when using standalone components.

So the benefits for beginning Angular developers are clearly there, but for more experienced developers the new feature is also easy to grasp as the standalone component looks pretty much like an ngModule. On top of this all, the new concept should not have any negative impact on performance and build-time. In fact, standalone components are expected to shorten the build-time somewhat.

All good news so far, are there any cons?

Standalone components certainly have their benefits for anyone picking up the Angular framework. Considering the projects that I've worked on, I clearly see benefits when it comes to small units such as directives and pipes. It will make life easier if you can just import one single directive or pipe. Small, representational components will be easier to share when they are standalone.

When developing more complex features, it is less obvious what standalone components bring us. It is still up to the developer to decide how big a component can grow. Another concern: introducing standalone components in a large, existing codebase will be a considerable undertaking, while mixing the two concepts is potentially confusing.

Finally, we do reduce some boilerplate in our projects when taking out the ngModule, but each and every standalone component will need to import modules like Angulars' CommonModule. Suggestions have been made to cut up the CommonModule so you only have to import what is needed, or that features like the `*ngIf` directive are imported by default. Currently there are no concrete plans by the Angular team in this direction.

conclusion

With standalone components, the Angular team will introduce a new way of working that will make the learning curve for beginning Angular developers less steep. This introduction will not be 'breaking' in any respect, but will co-exist peacefully with the existing module system. The benefits offered by standalone components will be mostly felt in clearly defined, smaller pieces of code. Migrating an existing codebase to standalone components, if wanted at all, may be a challenge.

BIO

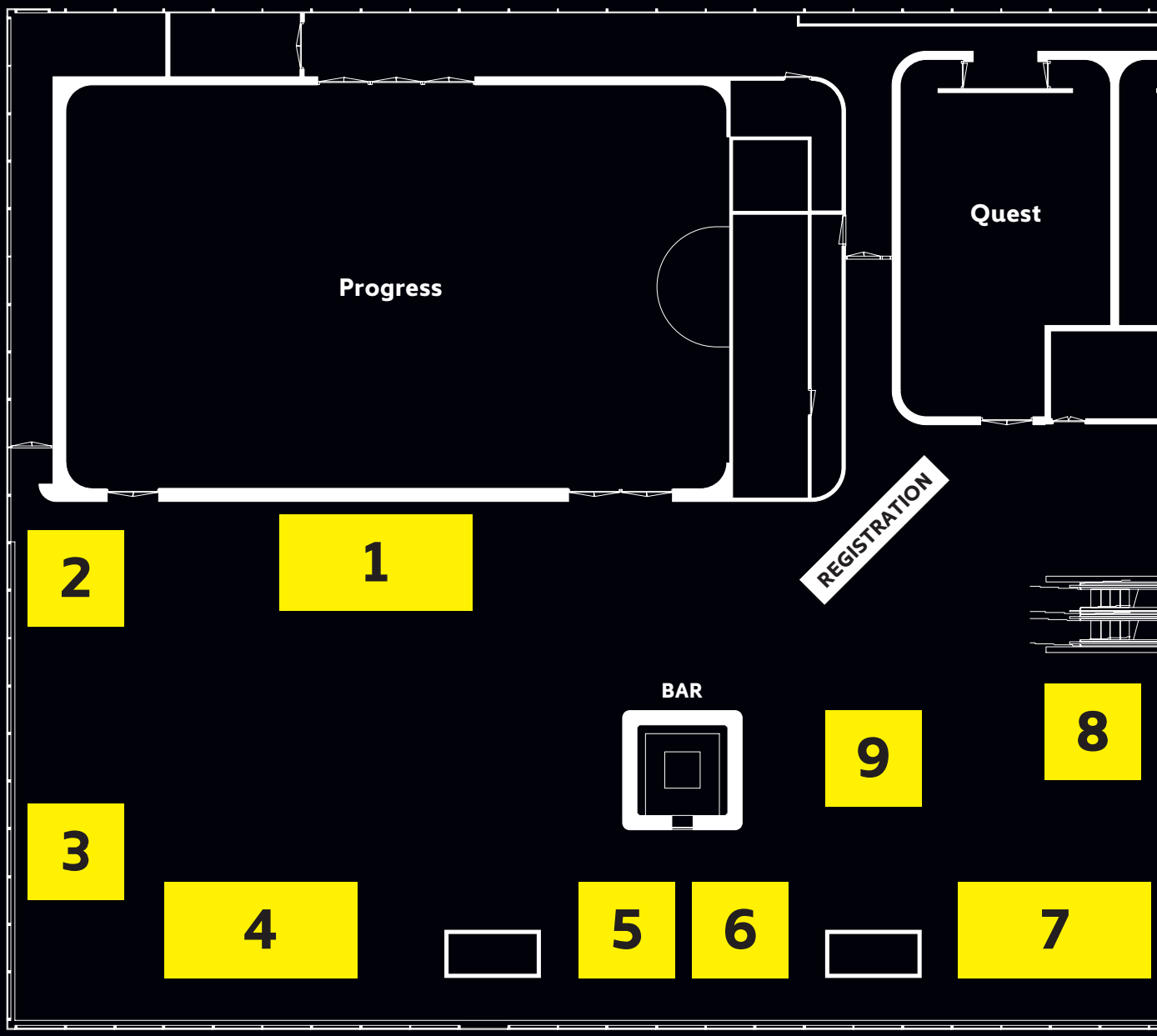
***Leonie van Rijn** is Frontend Developer at Rabobank Digital Platform and she plans to never stop learning and experimenting.*

Timetable Frontmania 2022

Time/Room	Progress	Mission 1	Mission 2	
9.30	Entrance open			
10:00 - 10:10	Opening			
10.10 - 10.40	Keynote: How to build an app in 20min **Bjørn Wikkeling**			
10.40 - 11:00	Transit zone: Coffee break			
11.00 - 11.45	Stop using JavaScript for that: moving features from JS to CSS and HTML **Kilian Valkhof**	Mobile App maintenance easy? Guess again **Wim Selles**	Build UI more confident using Component Testing **Horacio Herrera**	
12.00 - 12.45	Web performance APIs you (probably) didn't know existed **Matheus Albuquerque**	**Look on the website for more info!**	Maintaining a component library at scale **Joran Quinten**	
			Demystifying Web Accessibility **Josephine Schaefer**	
			Who is Jason Keays? How to talk about technical concepts to non-technical stakeholders **Jorrik Kleinsma**	
12.45 - 13.45	Transit zone: Lunch break			
13.45 - 14.30	Keynote: Confessions from an Impostor **Kyle Simpson**			
14.30 - 15.15	How to Create Pure CSS Games **Elad Shechter**	Beyond responsive design: new and future media queries **Kilian Valkhof**	Remix party, you're invited! **Dave Bitter**	
15.15 - 15.45	Transit zone: Coffee & tea break			
15.45 - 16.30	Container Queries: The next step towards a truly modular CSS **Maarten van Hoof**	Why you should test UIs with Storybook **Yann Braga**	Experience Accessibility: a different view of the world **Drazen Jankovic**	
16.45 - 17:30	Inside Fiber: the in-depth overview you wanted a TLDR for **Matheus Albuquerque**	Improving the frontend customer performance experience **Bart Kooijman**	TypeScript Yin-Yang - why it's just JavaScript, but with seat belts fastened **Emil van Galen**	
17.30 - 18.30	Transit zone: Drinks & Music			

	Expedition	Quest	Workshop area
	Build your apps 10 times faster with TRPC: How we migrate **Davy Engone**	Fast multi-platform apps with Flutter **Robbin Schepers**	How to build a web3 app: From frontend to smart contract **Jessy The**
	A journey in draft web api's **Sven van Straalen**	Why LIT is 🔥 **Lucien Immink**	
	Vite - The new kid on the block **Dennis Spierenburg & Sjoerd Valk**	Modernize your APIs with GraphQL **Sohan Maheshwar**	Build web components with Lit **Lars den Bakker, Willem de Vries & Pascal Schilp**
	Micro Frontends: The What, the Why and the How **Peter Eijgermans**	Create AR Face filters with the Chrome Face Detection API **Jorrik Klijnsma**	
	Look on the website for more info!	Will Git Be Around Forever? A List of Possible Successors **Hanno Embregts**	

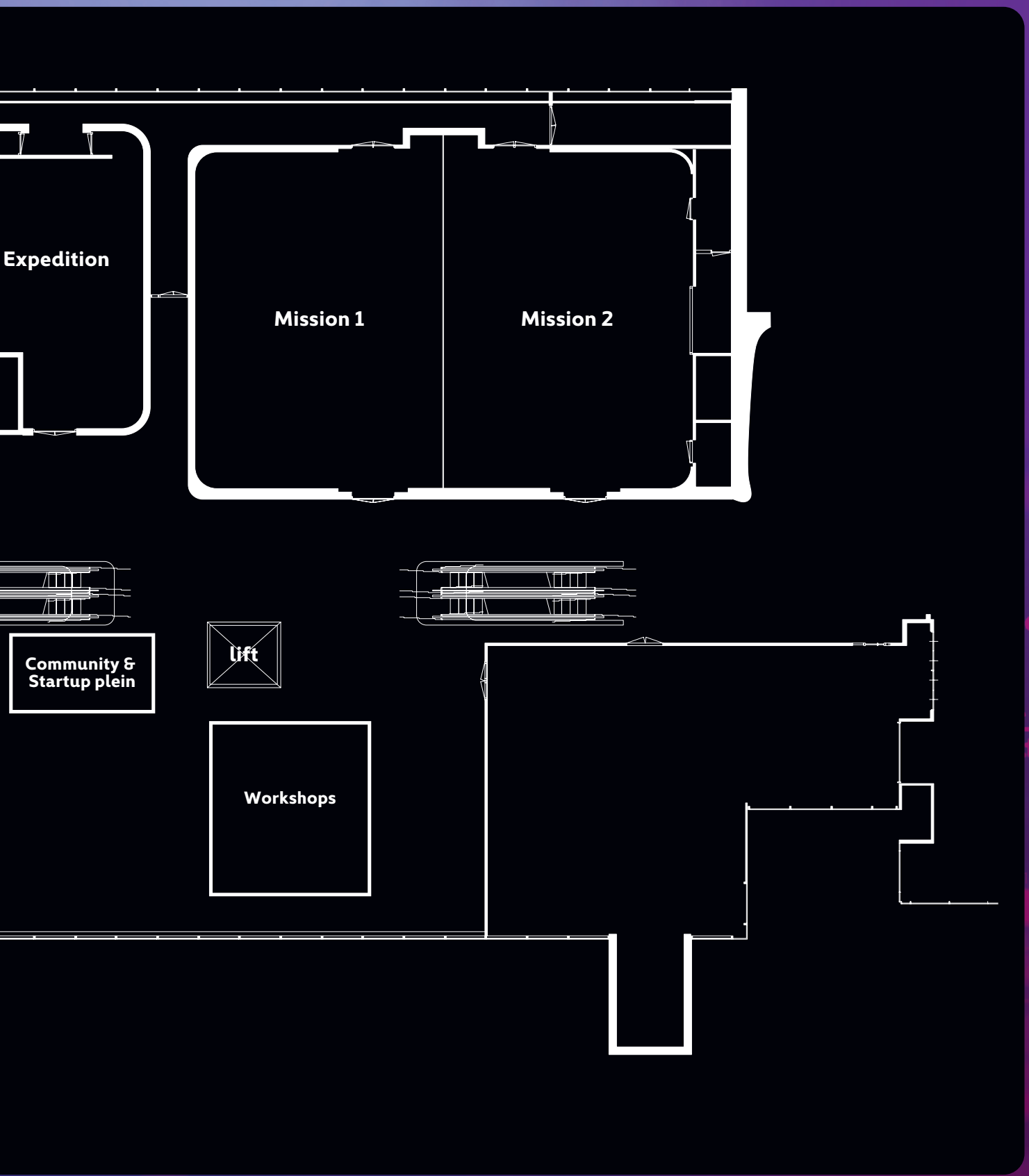
Floorplan Frontmania 2022



FRONT MANIA CONFERENCE 2022
October 5th

Rabobank	1	Saucelabs	7
Divotion	2	ABN AMRO	8
Pancompany	3	Sogeti	9
Software Bastards	4	ING	
Capgemini	5		
Ordina	6		

*Subject to change


Expedition
Mission 1
Mission 2
Community &
Startup plein
lift
Workshops


DJ
DEVITJOBS.NL
DE WEG NAAR EEN NIEUWE TECH JOB
Bezoek de site, solliciteer direct of meld je aan om de nieuwste vacatures te ontvangen.
devitjobs.nl/

FRONT MANIA CONFERENCE 2022

October 5th

FRONTMANIA thanks our sponsors & partners for making FRONTMANIA CONFERENCE 2022 possible!

As mentioned in the preface, we wouldn't be able to organize this awesome event without them

MAIN SPONSOR

Rabobank

MAIN PARTNER

CO-SPONSORS

SAUCELABS

PARTNERS

pan company
proven IT experts

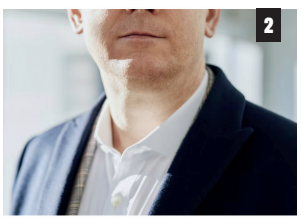
Smart cropping with native browser face detection

Many online services will help you with cropping an image while keeping face(s) in view. We can however do this just using an (experimental) browser native API. Let's build it!

The problem

Let's imagine that we need want to display an image on our webpage in an aspect ratio of 16 by 9. Now, this would be easy if we have a source image that has the same aspect ratio. But as a developer, you don't always have control over this. You've received the following image: **Image 1**.
Luckily, with modern CSS, we can easily make this 16 by 9 with the following to CSS lines:

```
img {
  aspect-ratio: 16 / 9;
  object-fit: cover;
}
```



Once doing this, we see the following result in the browser: **Image 2**. Oh no! We can't see the face anymore. This is because by default the object-position is set to center center which will center the crop on both the x-axis and the y-axis. No problem, we can just update our CSS to:

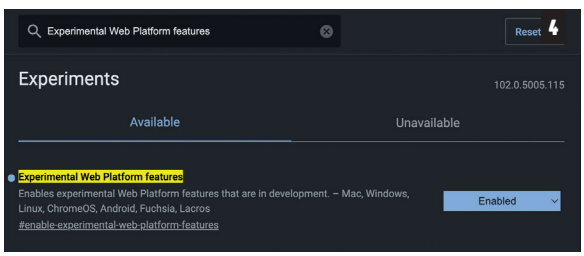
```
img {
  aspect-ratio: 16 / 9;
  object-fit: cover;
  object-postion: top center;
}
```

But wait, now the bottom of the face isn't visible: **Image 3**.

We can ultimately fix the object-position by passing pixel or percentage values to get the crop just right. This is however a painstaking process that you would have to do for all your images. Besides that, what if the images can be random and you can't cover all edge-cases? **We need a way to crop and position the crop just right for the face to be in view**.

The Face Detection API to the rescue!

The Face Detection API, a spinoff of the Barcode Detection API [1], is at the time of writing only available on Chrome after turning on a feature flag. You can do this by going to chrome://flags, searching for #enable-experimental-web-platform-features and turning it on. **Image 4**.



Your browser will restart and the Face Detection API will be available. So what can you do with this API? You can pass an image and it will return you the bounding box of the detected face(s). It will also give you an array of landmarks. These landmarks consist of detected eyes, noses and mouths per detected face. Today, we're just focussing on the actual faces. If we run the API on the previous image, we get the result of **listing 1**. So, we can now use this data to know where to position our crop, right? Well, that's what I thought. There is a pitfall. These values are referring to the intrinsic size of the image you pass. In our demo, we scaled down our image. Therefore, these values need to also be scaled down.

Mapping the intrinsically based values to the scaled-down based values

First, we need to know the width and the height of the intrinsic image. We can do this with the code of **listing 2**. Once we have the intrinsic width and height, we can map our face detection bound-

L1

```
[
  {
    "boundingBox": {
      "x": 230.64181518554688,
      "y": 208.36253356933594,
      "width": 292.52178955078125,
      "height": 292.52178955078125,
      "top": 208.36253356933594,
      "right": 523.1636047363281,
      "bottom": 500.8843231201172,
      "left": 230.64181518554688
    },
    "landmarks": [
      // removed from example
    ]
  }
]
```

L2

```
async #getIntrinsicImageNodeSize() {
    const that = this;

    return new Promise((resolve) => {
      var url = this.imageNode.src;
      var img = new Image();
      img.onload = function () {
        const { width, height } = img;

        that.intrinsicImageNodeSize = { width, height };
        resolve();
      };

      img.src = url;
    });
}
```

ing box as in **listing 3**. In essence, what we're doing is taking the face detection bounding box that is based on the intrinsic size of the image and calculating what the pixel values would be for our scaled-down image. Cool, we now have the following values:

```
{
  "top": 107.68089590146559,
  "bottom": 258.8549473488978,
  "left": 119.1947456238351,
  "right": 270.36880860493204,
  "width": 151.174062981097,
  "height": 151.1740514474322
}
```

We can now take these values to, for instance, draw a box on the detected face(s) for easier debugging: **Image 5**. If you want to learn more about this API, the Barcode Detection or Text Detection

5

API, you can watch my Friday Tip on A first look at the Shape Detection API [2]. **While you're there, make sure to subscribe!**

Using the face detected bounding box to set the crop position

Now that we have these values, we can set the object-position x and y value to be the top and left of the face detected bounding box. A downside to this, however, is that we need to recalculate these values everytime the image on the page resizes. A smarter way would be to set these value percentage based for both the debugging boxes and the actual object-position value. Let's set to percentages for the object-position property, see **listing 4**.

We basically use the top and left values of the face detected bounding box and calculate what percentage those values are based on the height and width of the image respectively. We can now see the smartly cropped face in view. **Image 6**. Great, that seems to work! Whichever image I use, I can be assured that the face will be in view. But what about multiple faces?

Creating a combined bounding box for multiple faces

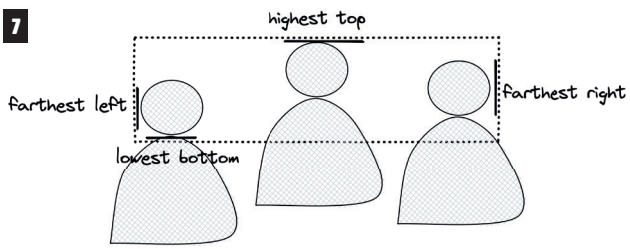
Luckily, we can quite easily implement the functionality to get the combined bounding box for the detected faces with the code of **listing 5**. Firstly, we get the highest top, lowest bottom, farthest left and farthest right of all the detected faces. We can then easily calculate what the width and height have to be for the combined bounding box. **Image 7**. Then, we use those values to set the value for the object-position just like we did earlier. A short demo for this image: **Image 8**. So, we detect all their faces: **Image 9**. And use the combined bounding box to smartly crop the image: **Image 10**.

6

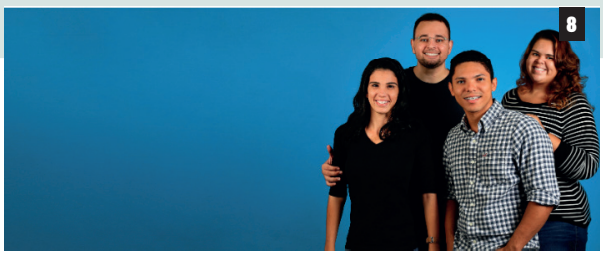
L3

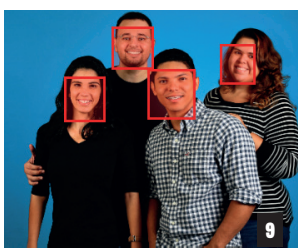
```
#getBoundingBoxForFace() {
    const imageNodeBoundingBox = this.imageNode.getBoundingClientRect();

    return {
      top:
        (faceDetectionBoundingBox.top / this.intrinsicImageNodeSize.height) *
        imageNodeBoundingBox.height,
      bottom:
        (faceDetectionBoundingBox.bottom / this.intrinsicImageNodeSize.height) *
        imageNodeBoundingBox.height,
      left:
        (faceDetectionBoundingBox.left / this.intrinsicImageNodeSize.width) *
        imageNodeBoundingBox.width,
      right:
        (faceDetectionBoundingBox.right / this.intrinsicImageNodeSize.width) *
        imageNodeBoundingBox.width,
      width:
        (faceDetectionBoundingBox.width / this.intrinsicImageNodeSize.width) *
        imageNodeBoundingBox.width,
      height:
        (faceDetectionBoundingBox.height / this.intrinsicImageNodeSize.height) *
        imageNodeBoundingBox.height,
    };
}
```

7



8





9

10

looking back

Naturally, we can optimize this demo even further with, for instance, margins for the crop and more. Next to that, this is in no way a production-ready solution due to it just working on Chrome behind a feature flag.

More important is that this shows just how powerful the browser can be. By combining two techniques, the Face Detection API and the object properties combined with the aspect-ratio property in CSS, we create an incredibly powerful tool. You can view the demo with the end result with examples here [3]. You can view the code of this demo on my Github page [4].

References

1. https://developer.mozilla.org/en-US/docs/Web/API/Barcode_Detection_API
2. https://www.youtube.com/watch?v=ZnSTi3Wbs7g
3. http://face-cropper-demo.davebitter.com/
4. https://github.com/DaveBitter/face-cropper-demo

BIO

***Dave Bitter** is a Front-end developer and Developer advocate. As a Developer Advocate, he's continuously on the hunt to learn new techniques, tools and possibilities. He revels in sharing the wonderful world of the web with fellow developers.*

L4

```
#setObjectCrop() {
    const boundingBox = this.#getBoundingBoxForFace();
    const { width: imageNodeWidth, height: imageNodeHeight } =
      this.imageNode.getBoundingClientRect();

    const { top, right } =
      this.#mapFaceDetectionBoundingBoxFromIntrinsicSize(boundingBox);

    const { setObjectFit, setObjectPosition } = this.options;

    if (setObjectPosition) {
      this.imageNode.style.objectPosition = `${Math.floor(
        (right / imageNodeWidth) * 100
      )}% ${Math.floor((top / imageNodeHeight) * 100)}%`;
    }

    if (setObjectFit) {
      this.imageNode.style.objectFit = "cover";
      this.imageNode.style.aspectRatio = "16 / 9";
    }
}
```

L5

```
#getOuterBoundingBoxFromFaces() {
    const sortedFacesByTop = this.faces.sort(
      (a, b) => b.boundingBox.top - a.boundingBox.top
    );
    const sortedFacesByBottom = this.faces.sort(
      (a, b) => b.boundingBox.top - a.boundingBox.top
    );
    const sortedFacesByLeft = this.faces.sort(
      (a, b) => b.boundingBox.left - a.boundingBox.left
    );
    const sortedFacesByRight = this.faces.sort(
      (a, b) => b.boundingBox.right - a.boundingBox.right
    );

    const top = sortedFacesByTop.at(-1).boundingBox.top;
    const bottom = sortedFacesByBottom.at(0).boundingBox.bottom;
    const left = sortedFacesByLeft.at(-1).boundingBox.left;
    const right = sortedFacesByRight.at(0).boundingBox.right;

    return {
      top,
      bottom,
      left,
      right,
      width: right - left,
      height: bottom - top,
    };
}
```

web components are here to take over your frontend, here's why

Building frontends using frameworks is writing new deprecated code. That's my opinion after the last couple of years of building modern frontend applications using web components. Because when you boil all frameworks down to their core features, they all bring the same to the table: building reusable components, making HTML creation and manipulation easy, ensuring encapsulation and doing this in a performant way using (Virtual) DOM engines.

Sure, one developer might prefer working with Hooks over a class-based component system. Or one might enjoy working with the simplicity of Vue over the enterprise-ready approach that Angular takes. Though at the end of the day, they're all for the same purpose: allow developers to rapidly build interactive webpages using a well-thought-out component system. And that's why I advocate for using web components: a set of tools for building modern frontend applications with zero dependencies by default.

BIO

Wessel Loth is Tech Lead at Arcady and member of the Frontmania Magazine editorial committee.

web components 101

'Web components' are a term you might have heard more and more about in the last couple of years. What started as an obscure browser API is now a serious contender for frontend developers around the world when they pick a component development system for new applications. Big tech companies like Google, Adobe and Microsoft are cashing in the profits of using web components. Even in the Dutch institutions like ING, Rabobank and the Police are actively working on frontend applications based on them.

The use case for web components in reality is quite simple. Imagine this: you want to build a dialog using the `<dialog>` element, but want to add a bunch of custom code specific to your scenario. When investigating how to do this, you're left with two obvious solutions: use a framework you know and love, or go vanilla. Going vanilla is often perceived as a negative thing: writing spaghetti code without a proper component architecture plan, manually chaining together event listeners or even fall back to an old classic: jQuery.

But what if you could make a `<fancy-dialog>` which extends the existing native one with your custom functionality? And if it was based on a component class with proper lifecycle hooks just like the framework du jour? And work out of the box in any modern browser, with no NPM or build tooling required? That's where web components come in.

The term 'web components' does not actually refer to a singular technology, but rather a suite of browser standards. They allow you to create custom elements on the page, to live alongside the native elements you know and love. This is achieved by utilizing three powerful browser standards: 'Custom Elements', 'Shadow DOM' and 'HTML Templates'. Each of these three standards are a piece of the puzzle that makes web components as powerful as they are.

custom elements

Custom elements are the bread and butter of web components. Like the name implies, using this standard you can define your own elements on the page with a custom HTML-tag, like `<fancy-dialog>`. The only requirements: an element needs to be a class extending from `HTMLElement`, and it needs to have a name with two or more hyphenated words so the browser can make a distinction between native and custom elements. A basic custom element looks something like this:

```
class FancyDialog extends HTMLElement {
  constructor() {
    super();
    this.innerHTML = `<some html to render>`;
  }
  // Other implementation details like handling
events
}
customElements.define('fancy-dialog', FancyDialog);
```

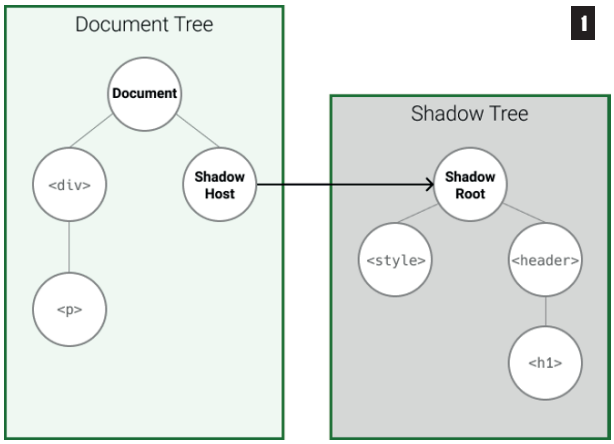
After execution, using the element on the page is as simple as placing a `<fancy-dialog>` tag anywhere on the page. The last line of code is the most critical one: it tells the custom element registry that the `FancyDialog` class should be instantiated when the defined tag is encountered. From that point on, the element behaves like any other native element present on the page. It's a node in the DOM tree, events bubble like you would expect and you can easily query or style them using the selector syntax you already know. And there's two added bonuses: you now can build components on the page similar to using components in any other framework, plus it all runs natively in the browser without hundreds of megabytes worth of toolchain in your node_modules directory.

THE TERM 'WEB COMPONENTS' DOES NOT ACTUALLY REFER TO A SINGULAR TECHNOLOGY, BUT RATHER A SUITE OF BROWSER STANDARDS.

shadow dom

Custom elements by themselves are cool, but rendering HTML using `this.innerHTML` isn't exactly doing anything for encapsulation. That's where the Shadow DOM comes in: possible the coolest sounding browser standard out there. To best describe the Shadow DOM, it's best to illustrate the concept with a diagram (see **Image 1**.)

On any node in the DOM, you can upgrade the node to become a shadow host. From that point on, you can interact with the newly created shadow root and treat it like you would with any other element on the page. You can use browser APIs like the query selector or appendChild to add content to the shadow DOM inside of the host node. The browser then takes the completed tree and



1

merges it into the Light DOM – the DOM tree you normally use when building a webpage. The added benefit: anything that happens inside of the Shadow DOM stays in the Shadow DOM.

You can add styling nodes inside of the Shadow DOM with extremely selectors such as `h1 { color: red; }` and the styles will only be applied within the separated tree structure. Events bubble like you would expect them to, with the added benefit of having more control on whether events can bubble outside of the shadow tree. The accessibility tree also works like it would normally: the shadow tree is flattened into the main DOM tree and assistive technologies can walk through the nodes like they normally would.

The nice thing is: you're already using the Shadow DOM. When you enable user agent Shadow DOM in your developer tools settings, you can actually inspect the implementation details of native elements such as the range input shown in **Image 2**.

```
▼<input type="range"> == $0
  ▼#shadow-root (user-agent)
    ▼<div>
      ▼<div pseudo="-webkit-slider-runnable-track" id="track">
          <div id="thumb"></div>
        </div>
      </div>
</input>
```

2

Encapsulation is all about hiding irrelevant implementation details from the outside, and making sure the inner workings of a component don't leak outside of its context. The Shadow DOM provides true encapsulation in a native way, without any of the weird workarounds that (Virtual) DOM engines use to create virtual encapsulation using tricks like CSS-class suffixing.

html templates

The only piece of the puzzle left for a robust and performant component system is a way to efficiently create and update HTML on the page. Luckily for us, there's a native way to easily achieve this goal using HTML Templates. Usage is simple: create a `<template>` element on the page, and add some other elements

and styles inside. When encountered by the parser on the page, it will parse and validate the contents, but does nothing with the element. Nothing is shown to the user, media URLs are not fetched and scripts are not executed. But when required, template elements can instantly be cloned and placed in parts of the DOM tree. Because the browser provides the low-level implementation for this behavior, it's extremely fast.

Together, these three standards form what we call 'web components'. When you combine these standards, you can build components using a class-based component system, you can easily truly encapsulate the inner workings of a component, and you can performantly manage HTML templates using the template element. And the browser support? It's fantastic! Because these are web standards, every major greenfield browser supports them. And fear not, even for legacy browsers there's a plethora of polyfills available to run web components smoothly

libraries & ecosystem

It's entirely possible to create web applications using just the native browser APIs. Using them makes me feel like I'm ten years old again: creating an index.html file, adding a script tag and boom! Stuff magically appears on the page, and it required minimal effort. But when you're looking at web components in the context of a company, it's not just a matter of finding a technology that works. There are all sorts of extra requirements, like the surrounding community & ecosystem, whether you can recruit developers familiar with the tech, and whether the developer experience is up to par with popular frameworks.

When it comes to the developer experience, it's entirely possible to create an easy-to-maintain codebase using just vanilla JavaScript. But you'll need to set up your own basic building blocks: figure out routing, your own way of managing styles on the page and set up your own build/bundling configuration. To make it a little bit easier, there's a handful of libraries available which take away the heavy lifting from you while still using browser standards under the hood. The most well-known libraries are:

- Lit
- Stencil
- Angular Elements

They all take a radically different approach. Lit is all about staying as close as possible to a vanilla custom element, while Stencil and Angular Elements are all about providing a batteries-included and enterprise-ready setup. At the end of the day though, all elements inherit from `HTMLElement` and fully interoperate with each other on the page. My personal approach when picking a technology is to start with vanilla, upgrade to Lit if necessary and only try the others if Lit does not cut it for you. The reason for this is that the Lit source code is simple to understand, there's zero build toolchain requirements and there's no additional dependencies you pull in.

Looking further at the surrounding ecosystem, there's some great developments happening in the web components sphere. Open Web Components (open-wc.org) has some great resources available for common set-ups and questions regarding tooling. Rocket (rocket.modern-web.dev) gives you a zero-JS server side rendered toolchain similar to Astro. Vaadin (vaadin.com/components) has many enterprise-grade components available for all your grid and chart needs. And even the ING bank has open-sourced all their core white-label components in Lion (github.com/ing-bank/lion) so you can easily add your own styles and have a flexible and fully accessible design system. These examples are just the tip of the iceberg: there's a pretty active community on Twitter continuously sharing cool new developments!

when (not) to use them

Looking at the ecosystem as a whole, it has grown sizably and became much more professional in the last few years. Finding out that even Photoshop in the browser was built using web components (a technical marvel regardless of technology choice) made me realize it's ready for production and a serious contender for all frontend applications.

There are some cons of course: it's hard to find experienced developers that have worked with web components. It can be jarring to have to write your own router when the ones available on NPM don't match your specific requirements. And depending on whether you picked a library or went full vanilla, you'll find few blogposts online helping you decide on best practices.

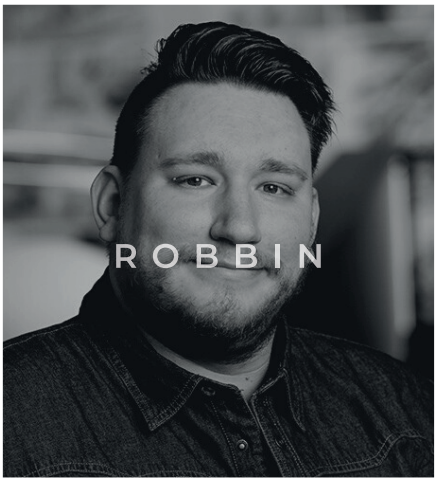
But when you look past those points, you'll find it's very easy nowadays to build a modern frontend application using just browser standards. Shipping complex applications with near-zero dependencies while maintaining full browser compatibility is nothing short of revolutionary. So, it's time to pull up your sleeves. Develop your frontend application using web components and free yourself from the eventual sunsetting of your favorite framework!

ORDINA JSROOTS DEVELOPERS

JORRIK'S FIRST LINES OF CODE WERE IN THE DAYS JQUERY RULED. THAT DIDN'T SCARE HIM.

NOW HE HAS 5+ YEARS OF FRONTEND EXPERIENCE USING REACT AND VUE DURING HIS DAY JOB. IN THE EVENING AND NIGHT HOURS, A LOT OF FUN PROJECTS AND OTHER LIBRARIES PASSED HIS 'GIT CLONE'.

WHEN NOT CODING HE GETS HIS FAIR SHARE OF LAUGHS GOING TO COMEDY SHOWS.



ROBBIN IS A PROGRAMMER IN CAREER, HOBBY AND PASSION. LIKE HE SAYS:
"I DREAM CODE, THINK IN LOGIC AND TALK FLUENT NERD."

ROBBIN HAS BEEN WRITING CODE SINCE HE WAS 11 AND HAS NEVER STOPPED.

MORE RECENTLY, ROBBIN HAS BEEN SPEAKING AND GIVING WORKSHOPS, MOSTLY ON FLUTTER AND SVELTE. FROM JAVASCRIPT TO COBOL AND FROM OS TO CLOUD, ROBBIN IS INTO IT.

PETER EIJGERMANS IS A LONG-TIME SOFTWARE DEVELOPER AND AN ADVENTUROUS AND PASSIONATE CODESMITH FRONTEND AT ORDINA NETHERLANDS.

HE LIKES TO TRAVEL AROUND THE WORLD WITH HIS BIKE. ALWAYS SEEKING FOR THE UNEXPECTED AND UNKNOWN. LIKEWISE, IN HIS JOB, HE LIKES TO WORK WITH THE LATEST TECHNIQUES AND FRAMEWORKS. AND HE LOVES TO SHARE HIS EXPERIENCE BY SPEAKING AT CONFERENCES ALL OVER THE WORLD.

COME SEE US!

ORDINA
Ahead of change


We think there is a place for everyone within the bank. Including employees with autism.
Discover IT & how Rabobank creates an inclusive culture at **rabobank.jobs**

Rabobank